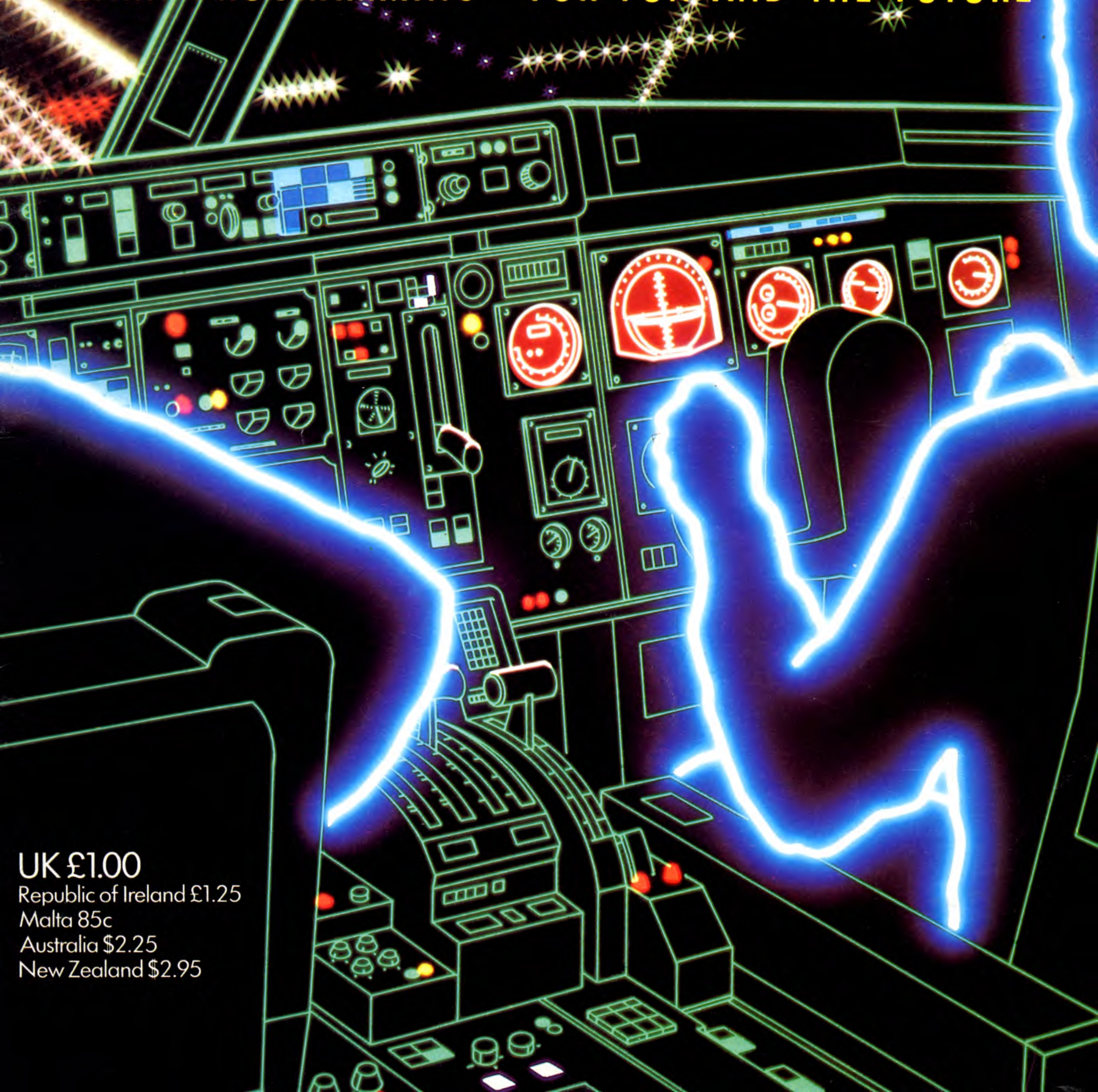
A MARSHALL CAVENDISH 23 COMPUTER COURSE IN WEEKLY PARTS

INPUT

LEARN PROGRAMMING - FOR FUN AND THE FUTURE



UK £1.00
Republic of Ireland £1.25
Malta 85c
Australia $2.25
New Zealand $2.95

INPUT

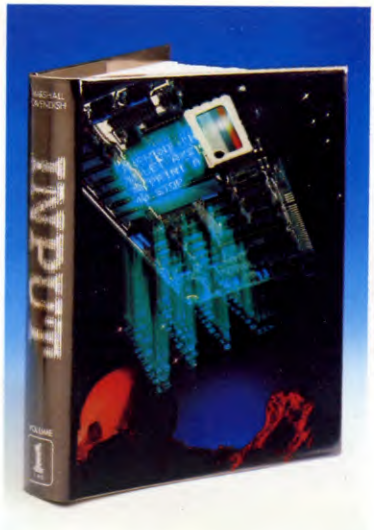
Vol. 2 **No 23**

INDEX

The last part of INPUT, Part 52, will contain a complete, cross-referenced index. For easy access to your growing collection, a cumulative index to the contents of each issue is contained on the inside back cover.

PICTURE CREDITS
Front cover, Paul Chave/Ian Stephen. Pages 701, 706, Phil Dobson. Pages 702, 703, 704, 705, 706, Berry Fallon Design. Page 709, Paul Chave/Spectrum. Page 710, Sporting Pictures. Page 712, Will Stephen. Page 715, Kevin O'Brien. Pages 716, 718, Paul Chave/Ian Stephen. Page 720, 727, Peter Reilly. Page 721, Grant Simon. Page 723, Grant Simon/Ian Stephen. Page 724, Tudor Artists. Pages 728, 730, Jeremy Gower.

Published by Marshall Cavendish Partworks Ltd, 58 Old Compton Street, London W1V 5PA, England. Printed by Artisan Presss, Leicester and Howard Hunt Litho, London.



There are four binders each holding 13 issues.

HOW TO ORDER YOUR BINDERS

UK and Republic of Ireland:
Send £4.95 (inc p & p) (IR£5.95) for each binder to the address below:
Marshall Cavendish Services Ltd, Department 980, Newtown Road, Hove, Sussex BN3 7DN

Australia: See inserts for details, or write to INPUT, Times Consultants, PO Box 213, Alexandria, NSW 2015

New Zealand: See inserts for details, or write to INPUT, Gordon and Gotch (NZ) Ltd, PO Box 1595, Wellington

Malta: Binders are available from local newsagents.

BACK NUMBERS

Back numbers are supplied at the regular cover price (subject to availability).

UK and Republic of Ireland:
INPUT, Dept AN, Marshall Cavendish Services, Newtown Road, Hove BN3 7DN

Australia, New Zealand and Malta:
Back numbers are available through your local newsagent.

COPIES BY POST

Our Subscription Department can supply copies to any UK address regularly at £1.00 each. For example the cost of 26 issues is £26.00; for any other quantity simply multiply the number of issues required by £1.00. Send your order, with payment to:

Subscription Department, Marshall Cavendish Services Ltd, Newtown Road, Hove, Sussex BN3 7DN

Please state the title of the publication and the part from which you wish to start.

HOW TO PAY: Readers in UK and Republic of Ireland: All cheques or postal orders for binders, back numbers and copies by post should be made payable to:
Marshall Cavendish Partworks Ltd.

QUERIES: When writing in, please give the make and model of your computer, as well as the Part No., page and line where the program is rejected or where it does not work. We can only answer specific queries – and please do not telephone. Send your queries to INPUT Queries, Marshall Cavendish Partworks Ltd, 58 Old Compton Street, London W1V 5PA.

INPUT IS SPECIALLY DESIGNED FOR:

The SINCLAIR ZX SPECTRUM (16K, 48K, 128 and +), COMMODORE 64 and 128, ACORN ELECTRON, BBC B and B+, and the DRAGON 32 and 64.

In addition, many of the programs and explanations are also suitable for the SINCLAIR ZX81, COMMODORE VIC 20, and TANDY COLOUR COMPUTER in 32K with extended BASIC. Programs and text which are specifically for particular machines are indicated by the following symbols:

SPECTRUM 16K, 48K, 128, and +
COMMODORE 64 and 128
ACORN ELECTRON, BBC B and B+
DRAGON 32 and 64
ZX81
VIC 20
TANDY TRS80 COLOUR COMPUTER

YOU HUM IT - I'LL PLAY IT

- MUSICAL NOTATION
- A MUSICAL RULER
- KEY SIGNATURES
- ADDING RHYTHM
- PLAYING A TUNE

The real beauty of computer music is that the machine plays it for you. Find out how to transcribe any written music into a program that remembers your favourite tunes

Music has two main elements—pitch and rhythm. The first article on music (pages 669 to 675) dealt mainly with pitch, which the computer provided subject to which keys you pressed, with you adding a touch of rhythm. This article not only adds a predetermined rhythm to the music that the computer plays, but also explains how you can turn any piece of sheet music into the DATA that your computer needs to play the written tunes.

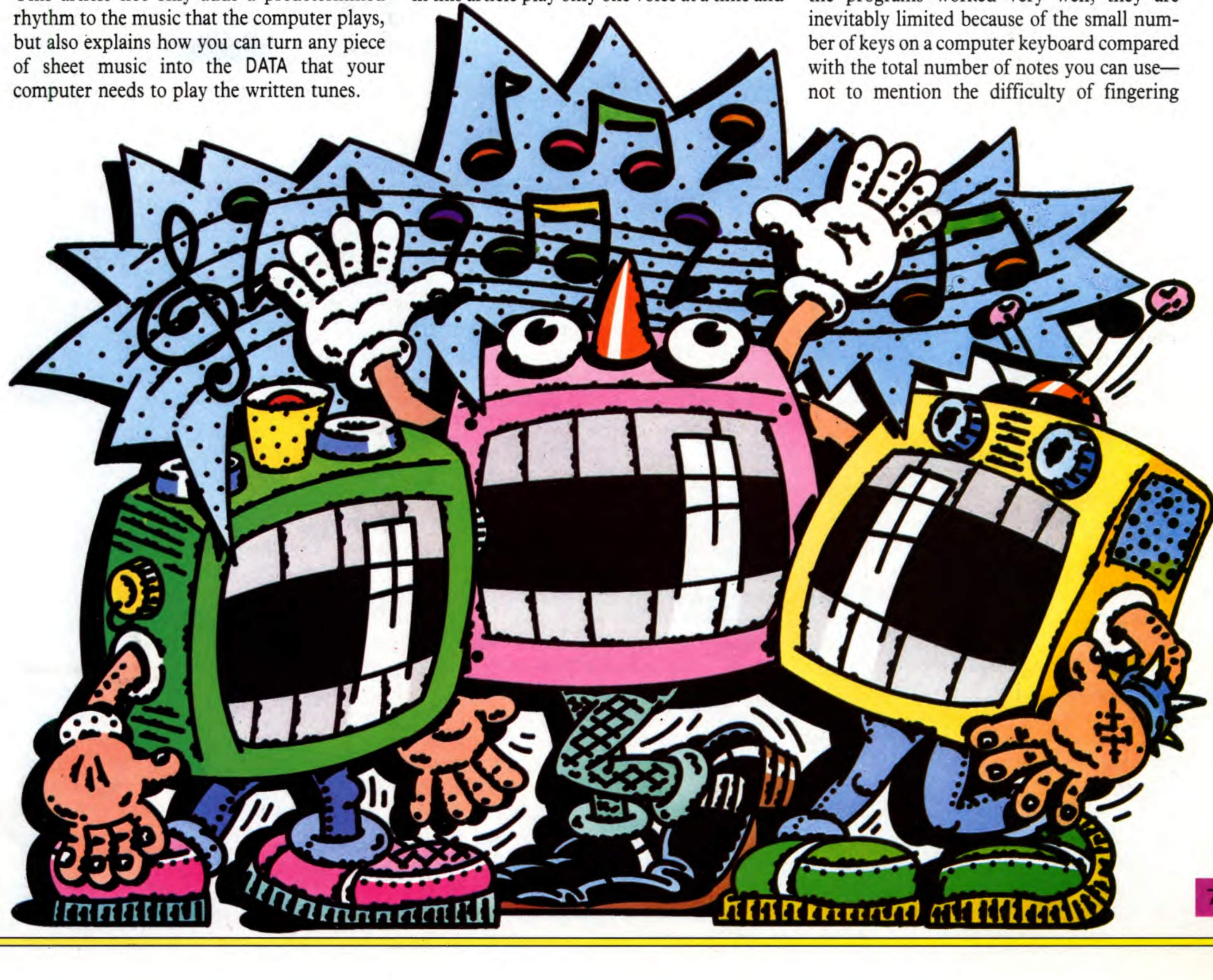
STANDARD MUSICAL NOTATION

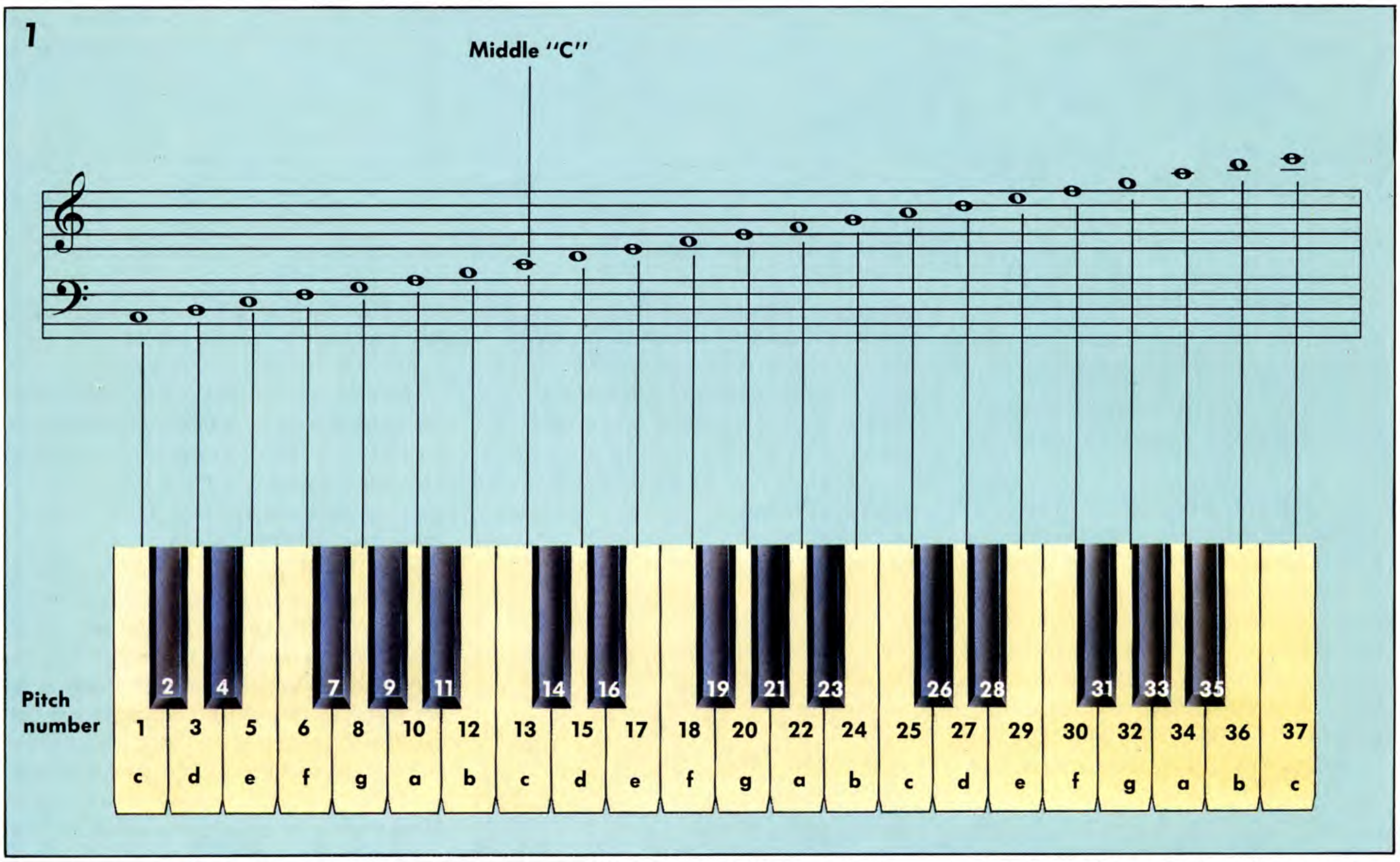
Music is usually written on one or two groups of five horizontal lines, called 'staves' (the singular is 'staff'). Various note symbols are placed on or between the lines of the staves, their height indicating the pitch, the order indicating the order of notes in the piece (they are played from left to right) and the shape of the symbols indicating the notes' durations.

Notes which line up vertically are played together to sound as chords, but the programs in this article play only one voice at a time and so cannot play chords. Vertical lines called 'bar lines' divide up the music horizontally into groups of notes called 'bars'; each bar has the same duration as any other bar, though each one may be internally divided into any number of notes, as long as its total duration is the same as other bars.

PITCH INFORMATION

The keyboard program in the last article turned the computers' keyboards into something resembling a musical instrument. While the programs worked very well, they are inevitably limited because of the small number of keys on a computer keyboard compared with the total number of notes you can use—not to mention the difficulty of fingering





them. But there is no reason why you should be confined to the same limits when you use tunes in your own programs.

You can quite easily convert any piece of written music into the numbers that your computer needs for each of these notes without needing very much knowledge of music: just use the 'musical ruler' described later on in this article. But, first, here is a brief explanation of some of the basics behind written music.

UNDERSTANDING THE NOTES

Fig. 1 shows a typical piece of written music. This would just play each note available on a keyboard in a rising succession—a 'chromatic' scale.

The large symbols at the extreme left of the staves are called 'clefs': the top one is called the 'treble clef' and the lower one the 'bass clef'. The clefs simply give the note symbols on the staves a pitch—without a clef, the pitch of any note on a staff is indeterminate.

Note symbols are placed either between two lines, or squarely over a line.

Notes above or below the staves are related to the staves by means of 'leger lines'; so the note Middle C lies on a leger line midway between the two staves, and the three highest notes in the diagram are on leger lines above the upper staff, at the righthand end.

It can be very confusing, and time-consuming, to translate each note on a sheet of music into the pitch and duration values for your computer, especially if you cannot read music. But there is an easy method.

If you look at the diagram, you can see that pitch values from 1 to 37 are marked beneath each note. These are the basis for the numbers you need in your program.

You could refer to this diagram whenever you wanted to 'transcribe' a tune, but even that would mean a lot of time spent comparing your sheet music to the diagram and finding the right note. A much easier method is to have a measure of some sort which you can place over any note on a musical staff, and just read off its pitch value for your computer.

A PITCH RULE

To make this, all you need to do is to mark a piece of paper with a musical scale. Luckily, this only takes a minute or so, although you probably need a different scale for each piece of music, as the size of the staves can vary.

First, place your paper over the staff, and copy onto it the staff lines showing the possible positions of a note. Then all you do is write the pitch number for each note against the corresponding mark. This ruler can trace pitch values directly off the sheet music.

The actual values for each pitch are given in the table and it's probably easier to use these for the Spectrum (where the numbers are simple) or the Dragon and Tandy (which use letters). The number for the Commodores and Acorns are more complicated so it's easier to use the general scale and then convert them in the program. To find out what to mark on each line, compare the numbers on the diagram to the table, and read off your computer's equivalent number.

SHARPS AND FLATS

Unless you convert only pieces of music written in C Major, you will at some time need to transcribe sharps and flats—and these do not have the same pitch as the natural note.

By looking at the key signature, you can see what, if any, flats or sharps the tune uses (how to understand key signatures is explained later in this article). Any sharps or flats mean you need to change the values on your ruler.

Sharps and flats are the black keys on the drawing of the keyboard, and their pitch numbers are provided. So, for example, the pitch number of the lowest C sharp is 2 (using the general scale of pitch numbers in the table, you can see what this is for your micro). This is the same as the pitch number for D flat, and is between 1 and 3, the pitch numbers for C and D.

But sharps and flats are not shown on the

General scale	Spectrum	Commodore High byte	Commodore Low byte	Vic	Acorn	Dragon/ Tandy
1	−12	8	97	192	5	C
2	−11	8	225	197	9	C+
3	−10	9	104	200	13	D
4	−9	9	247	203	17	D+
5	−8	10	143	206	21	E
6	−7	11	48	208	25	F
7	−6	11	218	211	29	F+
8	−5	12	143	214	33	G
9	−4	13	78	216	37	G+
10	−3	14	24	218	41	A
11	−2	14	239	220	45	A+
12	−1	15	210	222	49	B
13	0	16	195	224	53	C
14	1	17	195	226	57	C+
15	2	18	209	227	61	D
16	3	19	239	229	65	D+
17	4	21	31	231	69	E
18	5	22	96	232	73	F
19	6	23	181	233	77	F+
20	7	25	30	234	81	G
21	8	26	156	235	85	G+
22	9	28	49	236	89	A
23	10	29	223	237	93	A+
24	11	31	165	238	97	B
25	12	33	135	239	101	C
26	13	35	134	240	105	C+
27	14	37	162	N/A	109	D
28	15	39	223	,,	113	D+
29	16	42	62	,,	117	E
30	17	44	193	,,	121	F
31	18	47	107	,,	125	F+
32	19	50	60	,,	129	G
33	20	53	57	,,	133	G+
34	21	56	99	,,	137	A
35	22	59	190	,,	141	A+
36	23	63	75	,,	145	B
37	24	67	15	,,	149	C

staves in the diagram to prevent it from becoming too cluttered. A sharpened note is *notated* on a staff with its note symbol preceded by a 'sharp' sign, (which is like the computer 'hash' character); a flattened note appears on the staff preceded by a 'flat' sign, (which is like a lower case 'b').

Fig. 2 shows how the first few notes from middle C, including sharps and flats, are shown on the upper staff. Notice that while C sharp and D flat share the same pitch number and the same black key, they appear at different heights on the staff—since one is a C that has been raised, and the other is a D that has been lowered.

Individual notes in a piece may be sharpened or flattened, using the appropriate symbols immediately before the notes to be altered. A sharp or flat sign used like this is known as an *accidental*. Its effect lasts until the end of the bar it's used in, unless its effect is cancelled before this point. There's a third symbol, called a *natural*, which cancels a sharp or flat and restores the note to its unsharpened and un-flattened state. **Fig. 3** shows the first few notes of *Oh, I Do Like To Be Beside The Seaside*, including an 'E' sharpened by an accidental and naturalized a couple of notes later.

KEY SIGNATURES

Where the scale in which a piece is played contains a sharp or flat note, this note needs to be used instead of the natural every time. To signify that it is always a sharp or flat, the appropriate sign is marked at the beginning of the staff, after the clef. These marks are called the key signature. (You should note that the words 'key' and 'scale' mean the same, so the key of C Major means the same.)

The absence of a key signature indicates that the key is C Major, which is a kind of 'default' key, but a key signature of one or more sharps or flats means the piece is in a different key. In a key signature of one sharp, the sharp sign is at the position at which the note F is placed. Its effect is to sharpen all the Fs, in all octaves, throughout the piece.

You may remember from the earlier article that it's necessary to sharpen the Fs to obtain the arrangement of musical intervals for B Major. The key signature of one sharp achieves this, and it's known as the key signature of B Major. So if you're transcribing a piece that has some sharps or flats just to the right of the clefs, remember to modify all the Fs or Cs, or whatever, for the whole piece, and use the corrected pitch number.

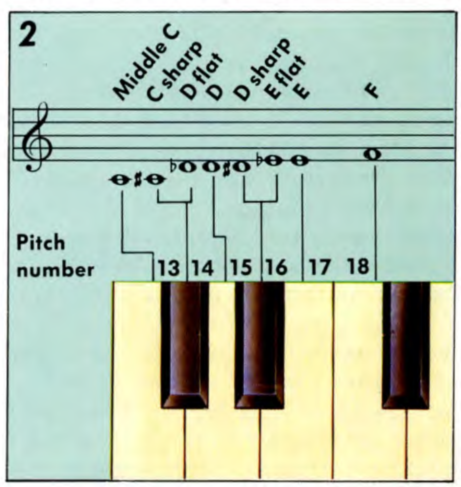
Fig. 4 shows the first few notes of *Pop Goes the Weasel* in the key of F Major, that is with a key signature of one flat. The pitch numbers are shown: notice that the fourth note is B flat, with a pitch number of 23 on the general scale. This is where your ruler comes in again. If your piece of music contains an F sharp, instead of writing the pitch number for F next to the F mark on your scale, you should write the number for F sharp.

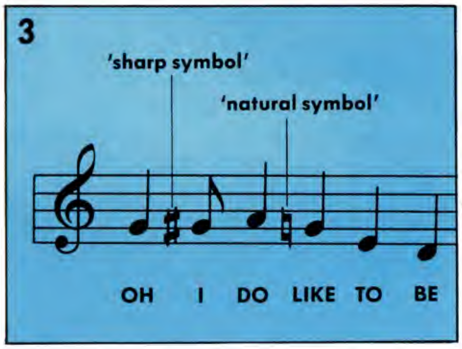
By doing this for any sharps or flats in the key signature, sharps and flats become just as easy to transcribe as natural notes.

You should be careful, though: sometimes you may come across accidental sharps, flats or naturals—these have priority over a key signature.

A MATTER OF TIMING

Once you've worked out the pitch of each note, the next thing is to work out its *duration*, or how long it sounds for. You'll also have to





note down the duration of *rest symbols* which indicate gaps, or periods of silence between notes. **Fig. 5** is a table that shows the note symbols and corresponding rest symbols, with their relative durations. It is a very good idea to use relative durations for notes and then choose a tempo which defines their *actual* durations at a later stage—that way, you can quickly and easily experiment to find the tempo which sounds best. You can see how this works in practise from the program further on in this article.

Both American and English names for the symbols are given: American names are more logical, though they will be less familiar to British readers. Generally, the *stems* on the notes may point upwards or downwards, whichever direction looks neater.

The two longest rest symbols are little rectangular blocks which lie above or below the middle line of the staff. Following a note or a rest by a dot multiplies its duration by one and a half; a doubly-dotted note has a duration one-and-three-quarters the duration of the undotted note.

The clef at the beginning of a piece of music is followed by the key signature, if one is present, then by the *time signature*. This consists of two figures placed one above the other. Essentially, the upper figure shows how many beats or pulses there are in each bar, and the lower figure shows the duration of each of these beats. So 4/4 indicates that there are 4 crochet beats per bar. Again, you need not fully understand the significance of this if you are just transcribing sheet music into DATA statements for the music programs, although you might like to use it as a guideline for choosing the tempo.

TRANSCRIBING A SIMPLE TUNE

You can now begin to transcribe a piece of written music into the numbers or letters that your computer uses to play the tune—both for pitch and duration.

Fig. 6 shows the rhyme, *Three Blind Mice*, written on a musical staff, along with the numbers and letters which represent the general pitch. To find the actual pitch numbers for each computer you can look at the table given earlier in the article. It also shows the relative duration values.

There are several points here about the way the music is written. Firstly, where three 'quavers' (the table shows what the notes are called), occur together, as in bars 9 and 11, for example, their stems have been joined together. Small groups of two, three, or four quavers and semi-quavers are often joined like this, to make it easier to read the music. But apart from that, the notes are exactly the same—the joins do not change the pitch or duration of the notes.

In bar 8, there is a similar join—but this time it does change the notes a bit. The join here is between just two notes and is a curved line instead of a straight one. What it means is that the notes should be joined together to sound like one, longer, note. The join is known as a *tie*. You can have ties joining any number of notes together. To convert them into computer data, you should add up the individual durations of the notes to give one long note.

PROGRAM DATA

The pitch and duration numbers you've worked out for the tune are put into DATA statements at the end of the program from where they are READ and interpreted as the program runs. The DATA for *Three Blind Mice* has been added to the programs in this way. There is enough DATA in each statement for two bars except on the Dragon and Tandy. Some convention of this type is sensible, as it lets you find the DATA for a given bar easily.

The total duration of each bar should be the same, 12 units in the example used. So, if you transcribe a piece of music and the rhythm doesn't sound right, check that each bar contains the same number of duration units, and correct any errors if you find them. The screen-editing facilities of each computer allow you to duplicate DATA statements containing bars that are repeated, which can save a lot of time.

TEMPO

The duration values give the relative duration of the notes, but they don't dictate its tempo, or absolute speed. When you RUN the program, you will be asked to provide a tempo number which controls this: a small number will make the tune play fast, a higher number makes it play slower. In fact, the number you INPUT is a sort of inverse tempo, since in music a large tempo number means a fast speed, not a slow one. An inverse tempo is used here as it is easier to program.

An important point about the way the computers operate applies to all the programs. Suppose one bar contains a single, long note, and another contains many short notes. In the second case the lines that READ and test the DATA will be executed more times than in the first case, and so the speed at which they are played will fall a little because of the extra operating time. Therefore the speed of execution of these lines should be as high as possible. This is why there is no test for the end of DATA.

A convention could be used in which a negative number, say, signalled the end of the tune; this would require an extra test in these crucial lines and would make the tune slightly uneven. So the program is just allowed to run out of data, when it stops with an error message, something that is not normally considered good practice.

Of course, when you use a tune in your own programs, you can use a FOR ... NEXT loop to READ the exact number of pieces of DATA, which would prevent the error occurring.

Here are the programs to play the tune *Three Blind Mice*:
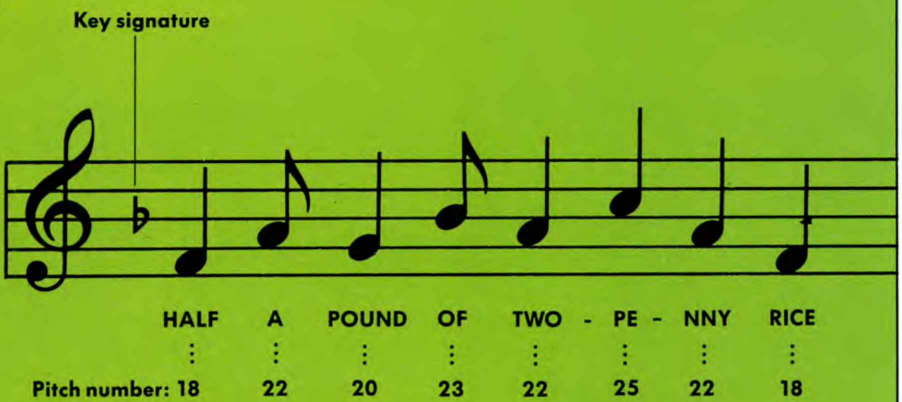
```
1Ø INPUT "ENTER TEMPO (1—5Ø)",t: LET
   t=t/1ØØ
2Ø IF t<Ø.ØØ1 OR t>Ø.5 THEN GOTO 1Ø
3Ø FOR n=Ø TO 1 STEP Ø: READ a,b: BEEP
   a*t,b: NEXT n
1ØØ DATA 6,4,6,2,12,Ø,6,4,6,2,12,Ø
11Ø DATA 6,7,4,5,2,5,12,4,6,7,4,5,2,5,1Ø,4,2,7
12Ø DATA 4,12,2,12,2,11,2,9,2,11
13Ø DATA 4,12,2,7,4,7,2,7
14Ø DATA 2,12,2,12,2,12,2,11,2,9,2,11
15Ø DATA 4,12,2,7,4,7,2,7
16Ø DATA 2,12,2,12,2,12,2,11,2,9,2,11
17Ø DATA 4,12,2,7,4,7,2,5
18Ø DATA 6,4,6,2,12,Ø
```

When you type in and RUN this program, you will be asked to INPUT a number between 1 and 50 for the tempo. As with the other programs, the value that you enter here is not actually the tempo, but the inverse tempo. What the computer does is to multiply the relative values for duration of each note by this tempo value—so, a small number gives you a fast speed, while a large number gives you a slow speed.

The reason for using inverse tempo is that it makes the programs simpler, and shorter. There is an IF ... THEN check to make sure you INPUT a reasonable value, in Line 2Ø.

Line 3Ø then actually plays the tune. Inside a FOR ... NEXT loop, it READs two pieces of DATA for each note—the computer's pitch value, and a relative duration value. Then, it

4



BEEPs before READing the next items of DATA.

When you use the routine for your own tunes, you can change the numbers in the FOR ... NEXT loop so that it READs the correct number of items of DATA. At the moment, the FOR ... NEXT loop has a STEP of Ø so it never reaches the end of the loop and only stops when the DATA has run out, with an E out of DATA message.

```
10 INPUT "TEMPO(30-50)";TP
20 GOSUB 3000
30 GOSUB 4000
100 READ P,D
110 IF P=0 THEN 130
120 POKE SE,EN: POKE SL,HQ%(P): POKE
    SI,LQ%(P)
130 FOR DL=1 TO D*TP: NEXT
140 POKE SE,EF
150 FOR DL=1 TO D*TP/3: NEXT
160 GOTO 100
3000 SI=54272
3010 SL=SI+1: SE=SI+4: EN=33:
    EF=32
3020 FOR I=SI TO SI+28: POKE I,0:NEXT I
3030 POKE SI+5,16*1+9
3040 POKE SI+6,16*15+9
3050 POKE SI+24,4
3060 RETURN
4000 DIM HQ%(37), LQ%(37)
4010 TMP=2227:P2=2↑(1/12)
4020 FOR I=1 TO 37
4030 LQ%(I)=TMP-256*INT(TMP/256):
    HQ%(I)=TMP/256
4040 TMP=TMP*P2
4050 NEXT: RETURN
10000 DATA 17,6,15,6,13,12
10002 DATA 17,6,15,6,13,12
10004 DATA 20,6,18,4,18,2,17,12
10006 DATA 20,6,18,4,18,2,17,10,20,2
10008 DATA 25,4,25,2,24,2,22,2,24,2,25,
    4,20,2,20,4,20,2
10010 DATA 25,2,25,2,25,2,24,2,22,2,24,
    2,25,4,20,2,20,2,20,2,20,2
10012 DATA 25,4,25,2,24,2,22,2,24,2,25,
    2,20,2,20,2,20,4,18,2
10014 DATA 17,6,15,6,13,12
```

This program uses pitch values from 1 to 37, and then changes them into SID chip values in the subroutines starting at Lines 3ØØØ and 4ØØØ. You could, if you wished, just use the correct DATA to start with and then miss out the calculation. This is, however, slightly more difficult to work out.

Line 1Ø lets you INPUT the tempo value: values of 30–50 are about right. The subroutine at 3ØØØ initializes the SID chip, and the one at 4ØØØ sets up arrays HQ% and LQ% for the 37 pitches, and fills them with the values for the high and low bytes which have to be POKEd into the SID chip. The loop from 1ØØ to 16Ø is responsible for READing and interpreting the DATA statements containing the music information at 1ØØØØ onwards.

Line 1ØØ reads the current pair of pitch and duration values into variables P and D. Line 12Ø switches on the envelope for voice 1, and POKEs the values for the specified pitch into the SID chip (variables SE, EN, SL and SI are used for speed: they are initialized in the subroutine at 3ØØØ). Line 11Ø tests the pitch and bypasses this line if the pitch is Ø, signifying the note is a rest.

Line 13Ø is a delay loop which gives the length of the note: the delay depends on the tempo and the duration value. Line 14Ø switches the envelope off, and Line 15Ø is another delay loop to give the 'breaks' between notes. The 'off' time given is a third of the length of the 'on' time. If the first value in this FOR ... NEXT loop is altered, the ratio of 'on' to 'off' time can be altered, making the notes more separate if the off time is lengthened or flow together more if the off time is shortened.

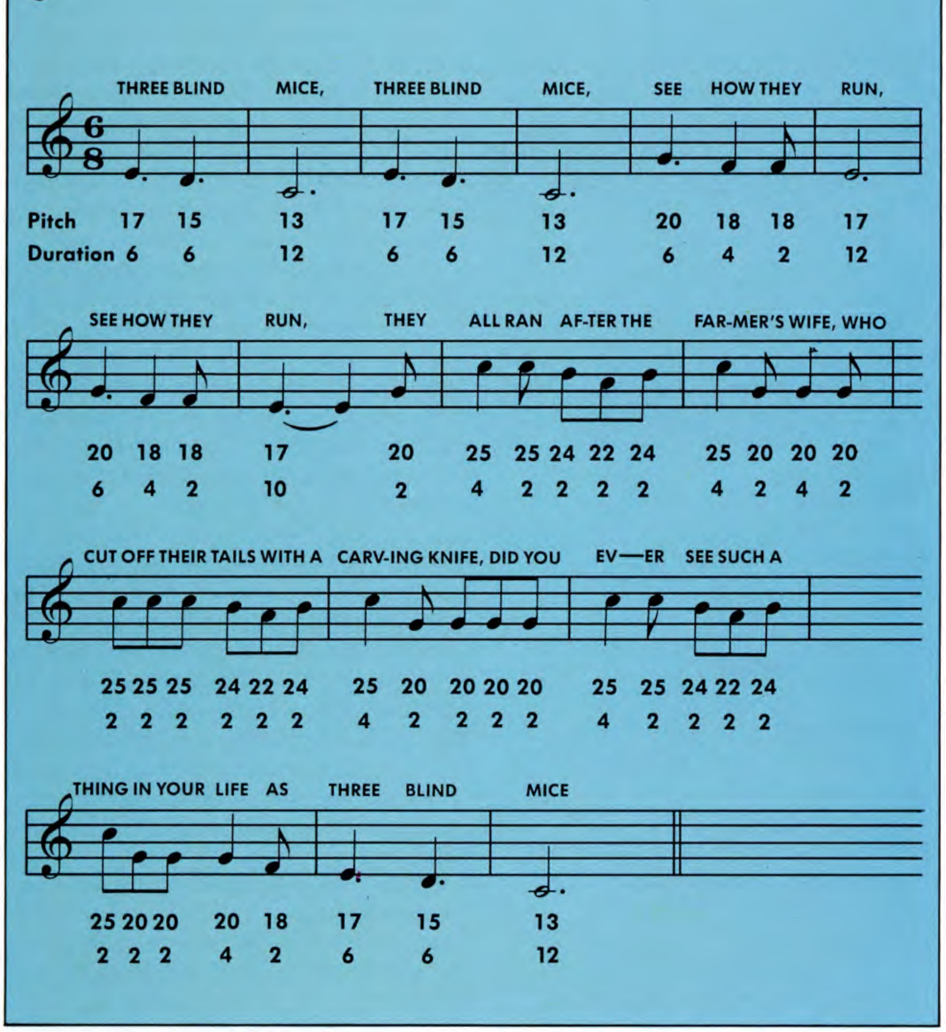
The upper limit of both FOR ... NEXT loops should be derived from D*TP to ensure that both the 'on' and 'off' times are related to the tempo and the duration parameter. The attack, decay, sustain and release values set in Lines 3Ø3Ø and 3Ø4Ø also affect the 'shape' of the notes which controls the quality of the sound.

```
10 INPUT "TEMPO(30-60)"; TP
20 GOSUB 4000
30 RV=36874:K0=0:K1=1
40 POKE 36878,4
100 READ P,D
```

5

Note symbol	European name	American name	Rest symbol	Relative duration	Duration if dotted
𝅝	Semibreve	Whole note	𝄻	16	24
𝅗𝅥	Minim	Half note	𝄼	8	12
♩	Crotchet	Quarter note	𝄽	4	6
♪	Quaver	Eighth note	𝄾	2	3
𝅘𝅥𝅯	Semiquaver	Sixteenth note	𝄿	1	1.5

6




```
110 IF P=K0 THEN 160
120 RG=RV
130 IF P>13 THEN P=P-12:
    RG=RG+K1
140 IF P>13 THEN P=P-12:
    RG=RG+K1
150 POKE RG,TA%(P)
160 FOR I=K0 TO TP*D:NEXT
170 POKE RG,K0
180 FOR I=K0 TO TP/3*D:NEXT
190 GOTO 100
4000 DIM TA%(37)
4011 FOR I=1 TO 13: READ
    V:TA%(I)=255-V: NEXT
4020 RETURN
4030 DATA 90,85,80,76,72,67,64,60,57,
    54,51,48,45
10000 DATA 17,6,15,6,13,12
10002 DATA 17,6,15,6,13,12
10004 DATA 20,6,18,4,18,2,17,12
10006 DATA 20,6,18,4,18,2,17,10,20,2
10008 DATA 25,4,25,2,24,2,22,2,24,2,25,
    4,20,2,20,4,20,2
10010 DATA 25,2,25,2,25,2,24,2,22,2,24,
    2,25,4,20,2,20,2,20,2,20,2
10012 DATA 25,4,25,2,24,2,22,2,24,2,25,
    2,20,2,20,2,20,4,18,2
10014 DATA 17,6,15,6,13,12
```

As in the Commodore 64 program, the first line reads in the tempo value, which should be between 30–60 for the tune given. The subroutine at 4000 initializes the array TA% to hold values to be POKEd into the VIC chip's sound registers.

The loop from 100 to 190 is responsible for READing and interpreting the DATA statements containing the music information at 10000 onwards. Line 100 READs the current pitch and duration values into variables P and D. Line 110 tests for a rest, and bypasses Lines 120–150, which handle the VIC chip values, if it detects one.

To obtain the 3 octave range, all three sound voices of the VIC chip are used: the same number POKEd into the three registers produces three different notes, each an octave above or below the others. So the three octaves of the range are each handled by a different VIC sound voice. This may seem awkward, but it enables you to produce better tuning than is otherwise possible.

The three pitch numbers in the range 1 to 37 are reduced to numbers in the range 1 to 13, and the correct pitch value is then taken from the array TA% and POKEd into one of the three sound-controlling registers. Lines 120–140 transform the pitch numbers to fall within this range and also select the correct sound-controlling register, whose address is left in the variable RE. (The register for the lowest octave is at 36874, the middle one is at 36875 and the upper one is at 36876). Line 160 contains a delay loop governing the note's 'on' time, and Line 180 is a loop controlling its 'off' time; the logic is similar to that of the Commodore 64 program above. Line 150 switches the note on by poking the correct value into the chosen VIC sound register, and Line 170 switches it off, after the first delay loop, by setting this register to 0.

The variables K0, K1, RV, RG are used to shorten the time taken to process the main loop, so as to achieve as regular a rhythm as possible.

```
10 INPUT"TEMPO(20-30)",TP
20 TP=TP/20
90 ENVELOPE1,1,0,0,0,0,0,0,30,
    -2,0,-5,100,50
95 ENVELOPE2,1,0,0,0,0,0,0,-127,
    -127,-127,-127,0,0
100 READ P,D
110 IF D*TP>254 THEN D=254/TP
120 IF P=0 THEN SOUND1,2,0,
    D*TP:GOTO 100
150 SOUND1,1,49+P*4,D*TP
```



```
2ØØ GOTO 1ØØ
1ØØØØ DATA 17,6,15,6,13,12
1ØØØ2 DATA 17,6,15,6,13,12
1ØØØ4 DATA 2Ø,6,18,4,18,2,17,12
1ØØØ6 DATA 2Ø,6,18,4,18,2,17,1Ø,2Ø,2
1ØØØ8 DATA 25,4,25,2,24,2,22,2,24,2,25,4,
   2Ø,2,2Ø,4,2Ø,2
1ØØ1Ø DATA 25,2,25,2,25,2,24,2,22,2,24,2,
   25,4,2Ø,2,2Ø,2,2Ø,2,2Ø,2
1ØØ12 DATA 25,4,25,2,24,2,22,2,24,2,25,2,
   2Ø,2,2Ø,2,2Ø,4,18,2
1ØØ14 DATA 17,6,15,6,13,12
```

The first line lets you INPUT the tempo; a value 20 to 30 is about right for the tune given.

Line 9Ø sets up ENVELOPE number 1, which the program uses for playing notes; it has an attack segment (attack rate is 3Ø), a decay segment (decay rate is −2), a sustain segment (where the volume remains steady) and a final release segment (release rate is −5). The peak volume is 100 units, and the sustain volume is 50 units. These values are provided by the last 6 parameters of the ENVELOPE command. Don't worry if you don't understand how the ENVELOPE command works, there'll be more on this in a later article.

Line 95 sets up ENVELOPE 2, which is used when the program is instructed to 'play' a rest. It's a kind of 'null' envelope, which doesn't actually let any sound through. The reason the program uses an ENVELOPE and a SOUND command to play a rest is so that the timing, whether of notes or of rests, is all taken care of by the system of queuing notes.

The rate at which notes are played is not governed by the execution time of the programs, but by the computer's internal queuing system. If you tried to produce rests with delay loops, the queuing system would handle all the notes and feed them out to the sound chip, but the delay loops would occur out of sequence and would effectively be lost. So the program uses a SOUND command both for notes which sound and for rests and the two kinds of event are then handled similarly by the queuing system.

Line 1ØØ reads in the current pitch and duration pair. Line 11Ø tests for a rest. If one is called for, it executes a rest of the required duration (D times tempo) using the 'null' envelope so no sound is made, and returns to Line 1ØØ. Line 15Ø plays an ordinary note. The expression 53+(P−1)*4 converts the pitch numbers 1–37 into the form expected by the SOUND command: 1 is converted into 53, 2 into 57, 3 into 61, and so on.

If you prefer, you need not perform this calculation on the DATA—by using the conversion table of pitch values earlier in this article, you could use the correct DATA to start with, instead.

```
1Ø INPUT"TEMPO (1−5Ø)□";TP
2Ø IF TP<1 THEN 1Ø
3Ø READ A$,D
4Ø PLAY"T"+STR$(INT(1+255/
   (TP*D)))+"O3"+A$
5Ø GOTO 3Ø
1ØØØ DATA E,6,D,6,C,12
1Ø1Ø DATA E,6,D,6,C,12
1Ø2Ø DATA G,6,F,4,F,2,E,12
1Ø3Ø DATA G,6,F,4,F,2,E,12
1Ø4Ø DATA O4C,4,O4C,2,B,2,A,2,B,2,
   O4C,4,G,2,G,4,G,2
1Ø5Ø DATA O4C,2,O4C,2,O4C,2,B,2,A,
   2,B,2,O4C,2,G,2,G,2,G,2,G,2
1Ø6Ø DATA O4C,4,O4C,2,B,2,A,2,B,2,
   O4C,2,G,2,G,2,G,4,F,2
1Ø7Ø DATA E,12,D,12,C,12
```

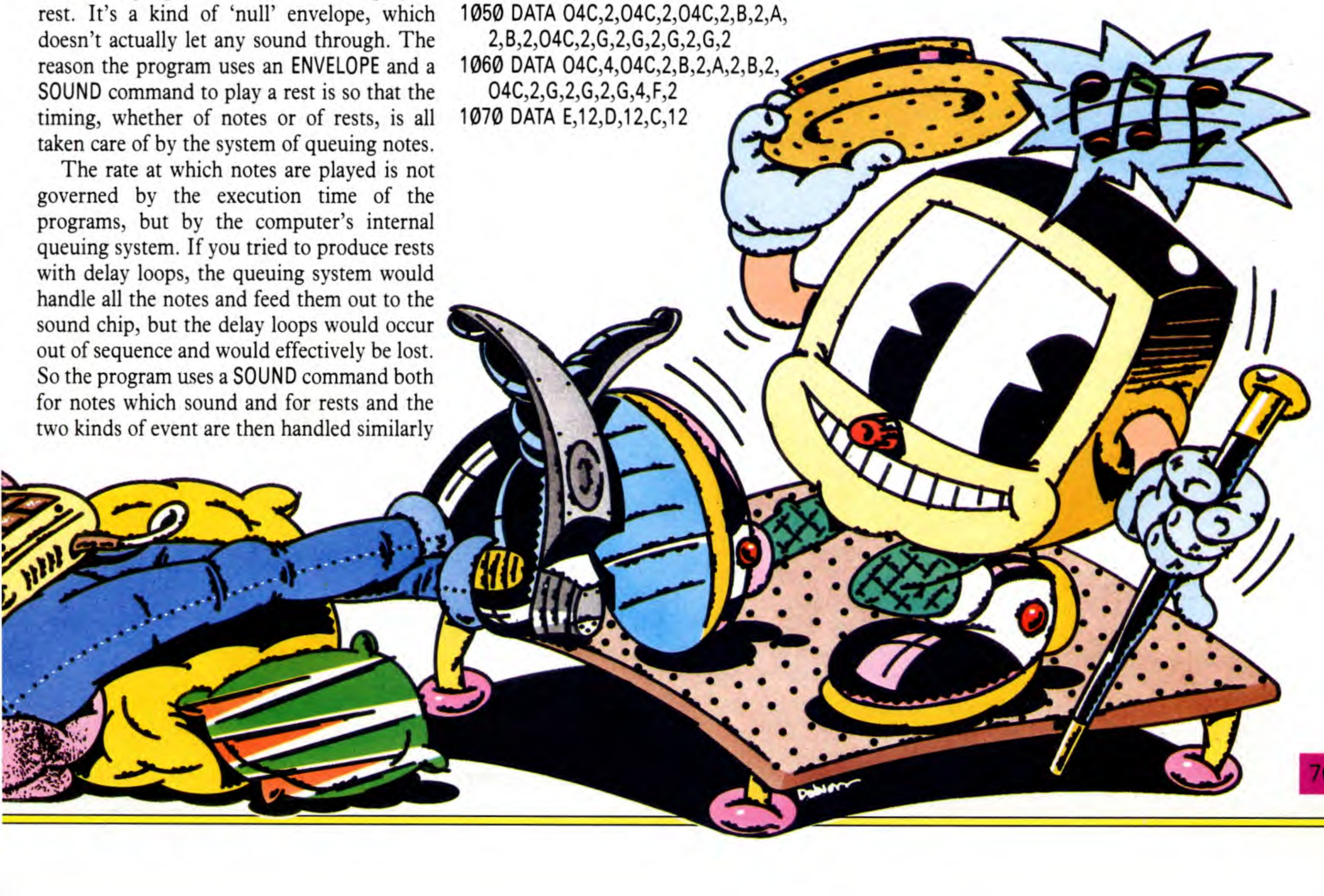
The Dragon and Tandy program begins by letting you INPUT a value which the program uses to determine the speed of the tune, its tempo. Line 2Ø checks to see whether the value you INPUT is less than 1, which is too small, and if it is, the computer returns to Line 1Ø for you to enter a new value.

Line 3Ø READs two variables—one for the pitch (the letter) and one for the duration of the note. To enter sharps add a plus or hash sign, and for flats a minus sign, after the note.

Then, in Line 4Ø, the computer plays the note, using the PLAY command. The T, in quotes, indicates that the next string in the line is a number between 1 and 255 (not 155 as indicated in the Dragon manual), which sets the actual tempo of the tune. The value you INPUT at the beginning of the program is inverted, and then converted into a string by the function STR$.

The "O3" after this in the same line sets the computer to octave 3, while the last part of the line (+A$) sets the pitch of the note.

Once the computer has PLAYed the note, it goes back to Line 3Ø, to READ the next note's DATA—unless, of course, the tune is over, in which case there will be no more DATA, and so the computer stops with an error message, as explained earlier in the article.



MORE SORTING METHODS

Here we look at how you can make small improvements to standard sort routines to increase their speed. And give details of one routine that beats the lot hands down!

All techniques used for sorting data into a predefined order come to grief because of time or memory restrictions. Some sort of compromise has to be established so that the most efficient sorting method can be used for a particular application. No single sorting method is ideal for every application.

We've already looked at some of the popular sorting techniques and here are a few more, some of which add only minor refinements but nevertheless return much better sorting speeds.

DELAYED REPLACEMENT

The exchange (bubble) sort has one significant handicap when more than a few items are to be sorted—it is extremely slow. But a very small improvement can be made to the standard algorithm to virtually halve the sorting time.

A good proportion of the time taken by the exchange sort is spent, on each pass, finding the highest incorrectly placed value and then comparing it in turn with each number next in line until a higher one is found. Exchanges take place many, many times—there's a lot of time-consuming shuffling along the line.

The delayed replacement sort works in a similar fashion but differs in that no exchange actually occurs until a pass has been completed. Let's look at an example, using a similar sequence of numbers to those used to illustrate the workings of the bubble sort on page 394:

top							**bottom**
67	35	72	19	47	38	11	86

The first value is 67. This is compared but not exchanged with 35 (as it would be in a standard bubble sort), nor is it with the first higher number it gets to—72. The latter is taken as the new high number and compared with 19, 47, 38 and 11 before resigning to the highest value in the sequence, 86. This just happens to be last and so remains in place. The list below shows the procedure step by step. Items enclosed are those identified as the highest incorrectly positioned value in each pass. They are subsequently exchanged with the value which occupies their slot. Thus, in the second row, value 72 is identified as the highest incorrectly placed value, exchanging with 11 by the third row.

top							**bottom**
67	35	72	19	47	38	11	[86]
67	35	[72]	19	47	38	[11]	86
[67]	35	11	19	47	[38]	72	86
38	35	11	19	[47]	67	72	86
[38]	35	11	[19]	47	67	72	86
19	[35]	[11]	38	47	67	72	86
[19]	[11]	35	38	47	67	72	86
11	19	35	38	47	67	72	86

The actual number of comparisons made is the same as with a standard bubble sort but the number of exchanges is very much less.

Key in the following program—a large part of it is exactly the same as appeared on page 395. Either make amendments to your existing program, or add Lines 4ØØØ onwards to a recording of the core program.

Be careful not to confuse capital I with the number 1 when keying in these programs! Acorn, Dragon and Tandy users should note the amendments to the core program—these follow the module listing.

You might also like to add the timing routine at Line 9Ø (page 395) to make comparisons.

```
1Ø POKE23658,8: LET T=Ø: INPUT
   "NUMBER OF ITEMS□";AA: IF AA<2
   THEN GOTO 1Ø
15 DIM A(AA)
2Ø PRINT:PRINT "UNSORTED TABLE": PRINT
3Ø FOR Z=1 TO AA
4Ø LET A(Z)=INT(RND*1ØØ)+1
5Ø PRINT TAB T; A(Z);:LET T=T+4: IF
   T>3Ø THEN LET T=Ø
6Ø NEXT Z
7Ø PRINT: PRINT: PRINT "PRESS S FOR
   SORT"
8Ø LET K$=INKEY$: IF K$<>"S" THEN
   GOTO 8Ø
9Ø GOSUB 4ØØØ
1ØØ PRINT: PRINT "SORTED TABLE": PRINT
11Ø LET T=Ø: FOR Z=1 TO AA
12Ø PRINT TAB T;A(Z);: LET T=T+4: IF
   T>3Ø THEN LET T=Ø
13Ø NEXT Z
14Ø GOTO 1Ø
3999 REM DELAYED REPLACEMENT SORT
4ØØØ FOR I=1 TO AA-1
4Ø1Ø LET K=I
4Ø2Ø FOR J=I+1 TO AA
4Ø3Ø IF A(J)<A(K) THEN LET K=J
4Ø4Ø NEXT J
4Ø5Ø IF I<>K THEN LET T=A(K): LET
   A(K)=A(I): LET A(I)=T
4Ø6Ø NEXT I
4Ø7Ø RETURN
```

```
1Ø PRINT: PRINT: INPUT "NUMBER OF
   ITEMS"; AA: IF AA<2 THEN 1Ø
15 DIM A(AA)
2Ø PRINT: PRINT "UNSORTED TABLE":PRINT
3Ø FOR Z=1 TO AA
4Ø A(Z)=INT(RND(1)*1ØØ)+1
5Ø PRINT A(Z),
6Ø NEXT Z
7Ø PRINT: PRINT: PRINT "PRESS S FOR
   SORT"
8Ø GET K$: IF K$<>"S" THEN 8Ø
9Ø GOSUB 4ØØØ
1ØØ PRINT: PRINT: PRINT "SORTED TABLE"
11Ø PRINT: FOR Z=1 TO AA
12Ø PRINT A(Z),
13Ø NEXT Z
14Ø RUN
3999 REM DELAYED REPLACEMENT SORT
4ØØØ FOR I=1 TO AA-1
4Ø1Ø K=I
4Ø2Ø FOR J=I+1 TO AA
4Ø3Ø IF A(J)<A(K) THEN K=J
4Ø4Ø NEXT J
4Ø5Ø IF I<>K THEN T=A(K): A(K)=A(I):
   A(I)=T
4Ø6Ø NEXT I
4Ø7Ø RETURN
```

To make the INPUT routines work, use a comma in place of a semicolon after the prompt in Line 1Ø and change Lines 4Ø and 8Ø:

```
1Ø INPUT ''"NUMBER OF ITEMS",AA: IF
   AA<2 THEN 1Ø
4Ø A(Z)=RND(1ØØ)
8Ø K$=GET$: IF K$<>"S"THEN8Ø
```

- DELAYED REPLACEMENT SORT —A REFINED BUBBLE SORT
- THE SCATTER SORT— COPYING WHAT YOU MIGHT DO BY HAND
- QUICK AND SIMPLE—THE CARD PLAYER'S OR INSERTION SORT
- THE QUICKSORT—FASTEST OF THE LOT, BUT AT A PRICE






```
40 A(Z)=RND(1ØØ)
5Ø PRINT A(Z);
8Ø K$=INKEY$: IF K$<>"S" THEN 8Ø
12Ø PRINT A(Z);
```

THE SCATTER SORT

The *scatter sort* is another of the routines which makes use of the speed of the exchange (bubble) sort when it is used for partially ordered lists. A subsidiary array is created for a preliminary 'rough' sort. Target first and last values are established initially and further items are added to the list only once this has been sorted.

Although this is a relatively speedy routine, the use of a separate storage array is a drain on available memory.

Note that you have to specify the maximum value. This is done in Line 5Ø1Ø and is taken to be 1ØØ with this example as that is the random number limit specified in Line 4Ø.

```
9Ø GOSUB 5ØØØ
4999 REM SCATTER SORT
5ØØØ DIM B(1.2*AA+3Ø)
5Ø1Ø FOR J=1 TO AA: LET
    K=INT(A(J)*AA/1ØØ)+1
5Ø2Ø IF B(K)=Ø THEN LET B(K)=A(J):
    NEXT J: GOTO 5Ø4Ø
5Ø3Ø LET K=K+1: GOTO 5Ø2Ø
5Ø4Ø LET J=1: FOR K=1 TO 1.2*AA+3Ø:
    IF B(K)=Ø THEN NEXT K: GOTO 5Ø6Ø
5Ø5Ø LET A(J)=B(K): LET J=J+1: NEXT K
5Ø6Ø FOR J=AA-1 TO 1 STEP -1: LET
    F=-1
5Ø7Ø FOR K=1 TO J
5Ø8Ø IF A(K)>A(K+1) THEN LET F=Ø: LET
    T=A(K): LET A(K)=A(K+1): LET
    A(K+1)=T
5Ø9Ø NEXT K: IF F=Ø THEN NEXT J:RETURN
```

```
9Ø GOSUB 5ØØØ
4999 REM *** SCATTER SORT ***
5ØØØ DIM B(1.2*AA+3Ø)
5Ø1Ø FOR J=1 TO AA: K=INT
    (A(J)*AA/1ØØ)+1
5Ø2Ø IF B(K)=Ø THEN B(K)=A(J): NEXT:
    GOTO 5Ø4Ø
5Ø3Ø K=K+1: GOTO 5Ø2Ø
5Ø4Ø J=1: FOR K=1 TO 1.2*AA+3Ø: IF
    B(K)=Ø THEN NEXT: GOTO 5Ø6Ø
5Ø5Ø A(J)=B(K): J=J+1: NEXT
5Ø6Ø FOR J=AA-2 TO 1 STEP -1:
    F=-1
5Ø7Ø FOR K=1 TO J+1
5Ø8Ø IF A(K)>A(K+1) THEN F=Ø:
    T=A(K): A(K)=A(K+1): A(K+1)=T
5Ø9Ø NEXT: IF F=Ø THEN NEXT:RETURN
```

The value 1.2 in Line 5ØØØ can be adjusted to provide sufficient room for the array. The scatter sort mimics, in some respects, how you might sort something yourself. Everything's laid out. The lowest and highest (start and finish) limits are set. And then everything is positioned according to priority.

INSERTION SORT

An improvement for many applications is the *insertion sort*. Lay out these playing cards left to right in front of you in the order shown:

Left				**Right**
9	4	5	7	2

The sort process starts from the left (or, in other words the first number to be sorted by a program). It looks for the first occasion when a *lower* number is out of order. A quick scan of the line shows that card 4 is out of place. So this is repositioned—inserted—before card 9 to give the new order:

Left				**Right**
4	9	5	7	2

Cards 4 and 9 are in order but then we come to 5, 7 and 2 which are not. In each instance, the card is picked out and inserted in its correct position, so at each subsequent pass the order of cards is as follows:

Left				Right
4	9	5	7	2
4	5	9	7	2
4	5	7	9	2
2	4	5	7	9

So you can see that there is no chopping up of groups or ordered, pair by pair comparison as with most other sorting methods. The sorted cards from left to right (effectively the numbers that are first to last in a program) proceed rightwards on each pass.

When you think about it, the process is not unlike that used by a card player when inspecting and then rearranging a hand of cards. In fact, the insertion sort is sometimes known as the *card player's sort* because of this procedure.

In the sort routine itself (see below), the sorted list is actually expanded downwards until the insertion position is found. Let's look at a slightly larger group of numbers and take a stage where some of these have already been sorted correctly:

Unsorted group	Sorted group
	34
	47
	59
	87
102〉〉〉〉〉〉〉〉	
26	144
73	167
193	

Value 102 is inserted in correct position and the sort routine proceeds in turn through the remaining values:

Unsorted group	Sorted group
26〉〉〉〉〉〉〉〉	
73	34
193	47
	59
	87
	102
	144
	167

And then the value 73 is inserted into the correct position leaving only one value unsorted:

	26
	34
	47
	59
73〉〉〉〉〉〉〉〉	
193	87
	102
	144
	167

And to complete the sort:

	26
	34
	47
	59
	73
	87
	102
	144
	167
193〉〉〉〉〉〉〉〉	

Add the following lines to the sorting demonstration program to compare the speed of this particular sorting routine:

```
90 GOSUB 6000
5999 REM INSERTION SORT
6000 FOR I=1 TO AA-1
6010 LET K=A(I+1)
6020 FOR J=I TO 1 STEP -1
6030 IF K>=A(J) THEN GOTO 6070
6040 LET A(J+1)=A(J)
6050 NEXT J
6060 LET J=0
6070 LET A(J+1)=K
6080 NEXT I: RETURN
```

```
90 GOSUB 6000
5999 REM *** INSERTION SORT ***
6000 FOR I=1 TO AA-1
6010 K=A(I+1)
6020 FOR J=I TO 1 STEP -1
6030 IF K>=A(J) THEN 6070
6040 A(J+1)=A(J)
6050 NEXT J
6060 J=0
6070 A(J+1)=K
6080 NEXT I: RETURN
```

The value of variable K is the next item to be compared from the unsorted list. The outer loop starting at Line 6000 scans the sorted list upwards from the lowest values to find the place where K has been inserted. Then the routine from Lines 6020 to 6050 'expands' the sorted list to make room for the new item. The program continues until the entire list of unsorted values has been scanned.

THE QUICKSORT

Fast though the insertion sort may be, it in no way compares with what is the technique preferred for many commercial programs: the *Quicksort*. The major drawbacks of Quicksort, though, are its programming complexity and memory requirements—so you don't often see it in programs specially designed for the limited memory of home computers.

But the Quicksort is unquestionably the fastest of the routines we've examined beyond a simple exchange sort comparison. It's well worth trying out! The algorithm of the Quicksort is fiendishly complex and impossible to explain simply. But basically, it uses one of two stacks and passes values to these depending on the value of that item when compared to a separate, arbitrary value.

```
90 GOSUB 7000
6999 REM QUICKSORT
7000 LET K=0: LET I=0: DIM S(AA)
7010 LET S(I+1)=1: LET S(I+2)=AA
7020 LET K=K+1
7030 IF K=0 THEN RETURN
7040 LET K=K-1: LET I=K+K
7050 LET A=S(I+1): LET B=S(I+2)
7060 LET Z=A(A): LET U=A: LET L=B+1
7070 LET L=L-1
7080 IF L=U THEN GOTO 7150
7090 IF Z<=A(L) THEN GOTO 7070
7100 LET A(U)=A(L)
7110 LET U=U+1
7120 IF L=U THEN GOTO 7150
7130 IF Z>=A(U) THEN GOTO 7110
7140 LET A(L)=A(U): GOTO 7070
7150 LET A(U)=Z
7160 IF B-U>=2 THEN LET I=K+K: LET
   S(I+1)=U+1: LET S(I+2)=B: LET
   K=K+1
7170 IF L-A>=2 THEN LET I=K+K: LET
   S(I+1)=A: LET S(I+2)=L-1: LET
   K=K+1
7180 GOTO 7030
```

```
90 GOSUB 7000
6999 REM *** QUICKSORT ***
7000 K=0: I=0: DIM S(AA)
7010 S(I+1)=1: S(I+2)=AA
7020 K=K+1
7030 IF K=0 THEN RETURN
7040 K=K-1: I=K+K
7050 A=S(I+1): B=S(I+2)
7060 Z=A(A): U=A: L=B+1
7070 L=L-1
7080 IF L=U THEN 7150
7090 IF Z<=A(L) THEN 7070
7100 A(U)=A(L)
7110 U=U+1
7120 IF L=U THEN 7150
7130 IF Z>=A(U) THEN 7110
7140 A(L)=A(U):GOTO 7070
7150 A(U)=Z
7160 IF B-U>=2 THEN I=K+K:
   S(I+1)=U+1: S(I+2)=B: K=K+1
7170 IF L-A>=2 THEN I=K+K:
   S(I+1)=A: S(I+2)=L-1: K=K+1
7180 GOTO 7030
```

ABOUT BULLETIN BOARDS

You too can share business and hobby information by linking your home computer by phone to any one of an ever-increasing number of computer-run noticeboards

If your phone bill can stand it, you can dial up the world through your computer and modem. On a more homely level, you can link up to fellow enthusiasts and join one of the booming areas of home computing—communication via *bulletin boards*.

A bulletin board is, at its simplest, just what it sounds like—the electronic equivalent of a club's notice board on which anyone can pin notices to be read by other members of the club. All you need to read what's on these bulletin boards and to write your own notices is a modem (see pages 618 to 621), some relatively simple terminal software, a telephone and, of course, the telephone number of a bulletin board. Access to bulletin boards is now relatively cheap—often the cost of a local phone call alone—and many modems are coming onto the market at very attractive prices.

Although bulletin boards, or 'BBs' as they are commonly known, have been around for a couple of years—especially in the USA—it's only recently that they have become popular in the UK. But there's every indication that BBs are set to mushroom. Already they are growing at the rate of several a month.

UPLOADING/DOWNLOADING

Computer owners with something in common—the same model of computer, for instance, or simply an interest in computer communications—can communicate with each other over long distances. BBs are not just used as simple notice boards. It is possible to upload and download software via a bulletin board. Many people are excited about the prospects for making superficially incompatible computers much more compatible by using these methods.

The principle behind this particular idea is that all characters are converted to their ASCII codes which are, of course, standard to the majority of computers. These in turn are converted into binary analogue signals.

In rather the same way that the dots and dashes of Morse Code represent the same characters in any language so the 'modulated' signals from a computer represent the same characters as far as just about every other computer is concerned. In the USA they are already talking about network software and hardware packages which will enable machines using the same operating system to upload and download software directly—no matter how different the machines.

The empty bulletin board upon which notices are pinned is the memory of a computer or, more usually, the computer's storage system. Obviously the storage system for a BB of any size must be at least a floppy disk drive. Anything smaller would simply be too small and would also probably be much too slow.

Anyone who has the correct equipment and software, knows the telephone number and, in some cases, the right password can read what's on the bulletin board and can write their own notices. What those who use a bulletin board are doing is reading from or writing to another computer's memory or storage system.

Most of the BBs springing up around the country have been set up by volunteers. In the tradition of grassroots computing, BBs are usually free—although there are those who charge a subscription. One of the problems, of course, is keeping non-members out. The usual security is a password, but if the huge mainframe computers of the US Defence Department cannot keep 'hackers' out then what chance has a small club of keeping out unwanted intruders. In fact, most BBs welcome intruders!

STANDARDS

Distance is no object, of course. One computer can communicate with another computer anywhere else in the world as long as they are using compatible modems. With the right equipment it is possible to gain access to more than a thousand BBs in the USA and Canada. Unfortunately you will need special equipment to get through to North America, since manufacturers do not conform to the European CCITT standard for modems.

CCITT is usually known in English as the Consultative Committee on International Telegraphy and Telephony but the real name of this United Nations agency is the Comité Consultatif International Téléphonique et Télégraphique. The organization sets standards for international communications. In many spheres, such as computers, this effectively means that standards are set for internal, domestic communications, too. This is welcome news for computer owners who consequently have the opportunity of communicating with other owners throughout Europe. In the UK, modems must also be



- USES OF BULLETIN BOARDS
- ABILITY TO UPLOAD AND DOWNLOAD
- COMMON STANDARDS
- BITS AND BAUD SPEEDS
- FULL OR HALF DUPLEX —WHICH WILL SUIT YOU?
- ACCESS BY MENUS
- EXPLORING THE NETWORK
- TERMINOLOGY USED

passed by the British Approvals Board for Telecommunications (BABT).

TRANSMISSION SPEED

International and national agencies still leave plenty of room for manoeuvre on the part of manufacturers. Modems can operate at different 'speeds', for example. The vast majority of BBs operate at 3ØØ baud which means that they send and receive data at 3ØØ bits per second—or one bit every 3.3 milliseconds. Most commercial systems such as Prestel or Micronet send and receive information at 1200/75 baud. This means that they send data at 1200 bits per second, but that the user can only transmit data at 75 bits per second.

Baud rates go up in steps: from 300 to 600, from 600 to 1200, 1200 to 2400 and so on, doubling every step up to 9600 baud.

On most computers, according to the CCITT standards for the RS232 interface, the voltages used to represent 0's and 1's are +12v and −12v. But some computers, such as the Commodore 64, for instance, use +5v



and 0v and a low cost converter may be necessary.

BITS AND BAUD RATES

Each byte is usually made up of a start bit, data bits, a *parity* bit for error checking and stop bit(s). Although there will be a natural logical sequence for the bits, it does not really matter how they are arranged—or even how many there are—as long as both the transmitting and receiving computers 'know' and 'agree' beforehand. Both computers must also be able to receive data at the speed at which it is being transmitted and to transmit data at the baud rate appropriate to the receiver.

There is a good reason why the BB baud rate should be that much slower than the baud rate used by the big commercial networks. BBs' resources are obviously much less than those of a big network company, and the equipment that's used would not seem out of place in the home of any micro owner. The slower the baud rate, the less chance there is of picking up unwanted signals. At fast baud rates, the slightest crackle could be mistaken for a bit. At slower baud rates, the pulse which indicates each bit can last much longer and can be much more well defined. This cuts down the chances of errors due to noise and reduces the need for sophisticated equipment.

Noise can sometimes be a big problem, and many are looking forward to the day when fibre optic cables become generally available. Fibre optics offer very 'clean' and very fast channels of communication.

DUPLEX

It's not just the baud rate that varies from one modem to another. A modem can be either *full duplex* or *half duplex*. Full duplex means that a modem can send and receive data at the same time, while half duplex means that a modem can only send or transmit at any one time—never both.

Half duplex is fine most of the time for home users, and may even be suitable for some business users. Unfortunately, mistakes can be time consuming and irritating. If you have asked the host computer to send you something, once it starts to transmit the data you must wait until the end of the transmission before being able to send a message to the host computer to say you have made a mistake! This can be very inconvenient for a business user.

BBs have been made possible by increasing sophistication in communications hardware and software. The big step forward as far as BBs are concerned was the introduction of auto-answering facilities on modems. Calls made to the bulletin board number are automatically answered and a line is then opened up between the two computers.

Microtip

When you're buying a modem, be certain that it is in fact suitable for use with your computer. Additional interfaces may be required, as may be leads and terminal software. Remember it is illegal to connect a modem to the public telephone network if it has not received official approval, an often long drawn out process which forces some manufacturers to anticipate approval but otherwise does not restrict sales. 'Approval pending' or similar such wording is a tell-tale sign that the unit is, at present, illegal to use.

Bear in mind that no model made up from a kit of parts will ever receive the necessary official approval.

If you're an absolute newcomer, seek advice from a knowledgeable dealer or approach the modem manufacturer direct. Also check with fellow enthusiasts what they are using. Try to avoid the same teething problems they had when setting up. The biggest of these teething problems will be actually getting used to the procedures involved in using a modem and establishing contact.

Terminal software falls into two categories—*dumb* and *smart*. Smart software will enable you to download software and save messages from other computers—essential if you want to operate a bulletin board—while dumb software lets you communicate with a BB but does not enable you to save software or messages from the BB.

The central function of the program that enables your micro to act as a terminal is to send any characters that are typed on the keyboard to the appropriate port on the computer and so send data received through the same port to the screen. Each computer is told, by this communications program, how to send data out and what to do with data coming in.

This sort of software is relatively simple, and even the sort of software that enables such functions as auto-answer is not particularly complex. In this instance, the program would simply look for an incoming message in the same way that other programs might look for a keypress.

ACCESS BY MENUS

Many people are puzzled to discover that one model of computer can apparently understand another completely different model. After all, computer manufacturers are constantly being criticized for failing to adopt common standards that would make communicating between computers easier. It sometimes seems, for instance, that one computer is able to send the other computer commands. This is not really so. Most BBs enable the user to find what they are looking for by using menus.

Within these menus, you are asked to choose an option by pressing the appropriate key or typing in the appropriate word. Superficially, this sometimes looks like one computer sending a command to another. But all that's happening is that the program in the host computer or in the modem looks for the characters sent by the guest computer. The ASCII code is used to transmit the characters so each one needs no more than seven bits.

It should be remembered that you are communicating with just one central computer. It's really this host computer that does most of the work.

EXPLORING BULLETIN BOARDS

The first step towards exploring the new and ever growing world of BBs is the purchase of a modem. The vast majority of BBs operate at 300 baud and most in Europe adhere to the CCITT standards. Access to the many BBs in North America is only possible through a special Bell standard modem. Prestel and other commercial networks operate on a baud rate of 1200/75, but there are switchable modems which offer a number of baud rates.

The second step will be to purchase or write the terminal software necessary for your computer to send and receive ASCII codes through a serial interface. The software usually comes with the modem, but be careful over your choice of dumb or smart software.

The third step is to find out the telephone numbers and technical details of BBs. These are printed in many of the computer magazines. BBs often show the telephone numbers of other BBs and there is an organization called AFPAS—Association of Free Public Access Systems.

After running the terminal software you will be able to call up and communicate with a bulletin board. If your modem has an auto-dial facility you will probably be connected automatically. If not you must dial the number yourself, listen for the *carrier tone* and then place the handset in the rubber cups on the modem in the case of an acoustic coupler

or, in the case of a direct-connect modem, flick the appropriate switch.

Note that some BBs require you to dial the number, let the phone ring once and then redial. This is so that the appropriate connection can be made.

Most BBs operate in a very similar way, so the procedure is roughly the same for each one. You should first of all see a greeting and a request for information displayed on your screen. This is the stage at which passwords and identification numbers are needed if the BB is one that requires a subscription. Some such BBs will not simply dismiss you. They may have special facilities for guests, information about the service and a limited amount of time and space on a special BB.

Free BBs will usually ask you for your name and home town and some technical specifications of your computer, the screen size, for example. The technical details are necessary to enable the host to set up the system to function properly with your computer.

You will then be given some technical information—the times during which the BB is available and a time limit for your call, perhaps. It is important to take note of the times during which the BB operates, because many are run on a purely voluntary basis and calling outside those times—especially in the middle of the night—might cause a lot of inconvenience.

Q+A

Is there any way I can protect messages I have left on a bulletin board?

The answer is, in theory, yes, in practice usually not. In some instances it is *very* easy to bypass preliminary password controls.

The very nature of a bulletin board is that it is open to public inspection—it is, in effect, a notice board which, almost by definition, means that information placed on it is intended for public scrutiny. But writing to it usually does mean password acceptance.

Some bulletin boards delve into areas which perhaps may be more closely linked to *electronic mail*—essentially private communications between individuals at personal or business level. Here, the use of passwords is very much the norm. Try to choose a novel password.



Like most viewdata services BBs are menu driven. Calling up one of the options on the main menu will probably lead you to another menu, which may in turn lead you to another menu, and so on until you eventually arrive at the option you require. Many BBs have a 'new user' section and that will be indicated on the main menu. The 'new user' section enables you to register with the BB. You are asked for your name and address and you can choose a password for use in future calls. Other options may include a technical information section, a 'Utilities' section which tells you the duration of your call so far and displays information about other services.

The heart of the BB is the actual bulletin board itself, which contains public notices which other people have put there for anyone to read. Often there is also a section for private messages. This really consists of smaller BBs to which access is limited by a password. Individuals can leave messages for other individuals, or groups can form their own bulletin board within the main bulletin board.

TERMINOLOGY

There are many technical terms used in computer communications which are not used very often in other branches of computing. This sometimes leads to confusion about the meaning of some terms. Here is an explanation of some of those terms which are likely to lead to confusion.

Teletext—Information services which involve broadcasting pages of information and displaying them using the 'spare' lines on a television set. The communication is one way only and users cannot transmit. Examples are CEEFAX, run by the BBC, and ORACLE, run by the independent television companies.

Viewdata—A similar service to Teletext but with two way communication possible. Prestel is an example of a Viewdata service. Communication is usually, though not necessarily, as many people think, by way of telephone lines.

Videotex—This is a generic term used to describe all forms of computer generated information services and covers both Teletext and Viewdata.

Bulletin boards also fall under the generic heading of Videotex as they are really a form of Viewdata. Some confusion arises because BBs have begun to develop and grow into something more than simple boards on which notices can be written and read. Some of them offer an information service.

Network—Strictly speaking, this refers to the direct linking up of a number of computers in a limited group for a specific purpose usually in a business or research environment. The main distinction between a network and other forms of computer communication is that a network involves direct connections between computers and/or peripherals. The purposes of networking computers is so that they can share facilities such as databases or printers rather than pool resources.

The meaning of the word network is, however, gradually being degraded and it now sometimes refers to any grouping of computers even if the connection between them is a central computer as in Viewdata.

ON-SCREEN FLIGHT SIMULATOR

This flight simulation program is similar to those used in flying schools to teach pilots how to fly using their instruments alone: the first part reproduces the cockpit

Games programs vary from sheer fantasy, which involves you entering an imaginary world and taking part in an adventure, to simulations of real-life situations. These allow you to test your capabilities in a potentially dangerous situation without causing harm to yourself or writing off millions of pounds worth of equipment.

Flight simulation programs have an element of fantasy—alone in the cockpit, all the crew stricken by a mysterious illness, you, single-handed, bring the aeroplane in to land. But sophisticated programs of this type have real practical use—so much so that most major airline companies and flying schools use them regularly.

TRAINING SIMULATORS

At the top end of the scale there is total simulation—'Phase 3' in the jargon of the Federal Aviation Administration (America's civil aviation governing body), which allows you to experience everything that the pilot in a real aeroplane does. You see what he sees through the cockpit window (including a slightly differently angled view for the co-pilot); you feel what he feels on take-off and landing, and during turbulence; and you hear what he hears, including air traffic control commands. Theoretically, a pilot can complete all his training in one of these, and obtain his licence without once leaving the ground.

TABLE-TOP SIMULATORS

At the other end of the scale, the flight simulation programs are very similar to the one that follows.

Table-top units that can be 'flown' in the classroom are good for teaching cockpit procedure, and developing the reflex speeds of the pilot.

They are essential for learning instrument flying, a technique that allows the pilot to navigate solely by referring to the instrument panel—something that every pilot has to do when the weather conditions are bad.

WHAT THE PROGRAM INVOLVES

The flight simulation program in this three part article assumes that you have taken over the controls when the aeroplane is 2000 metres in the air and 20,000 metres away from the target runway. Through the cockpit window you can see little—just the horizon, when it is in view, and the distant dot of the runway—so like a seasoned pilot you must rely on your expertise at responding to the instrument panel to bring you and all your passengers safely in to land.

THE INSTRUMENTS

There are four dials on your instrument panel. The first one tells you your airspeed. This varies according to whether you are diving (your speed increases), climbing (your speed falls off) and changing engine power. A counter underneath the airspeed dial tells you your compass bearing.

The second dial shows you where the horizon is in relation to your aeroplane. This means that even when the horizon is not in view through the cockpit window, you still know where it is. The counter underneath this dial gives you the bearing of the runway.

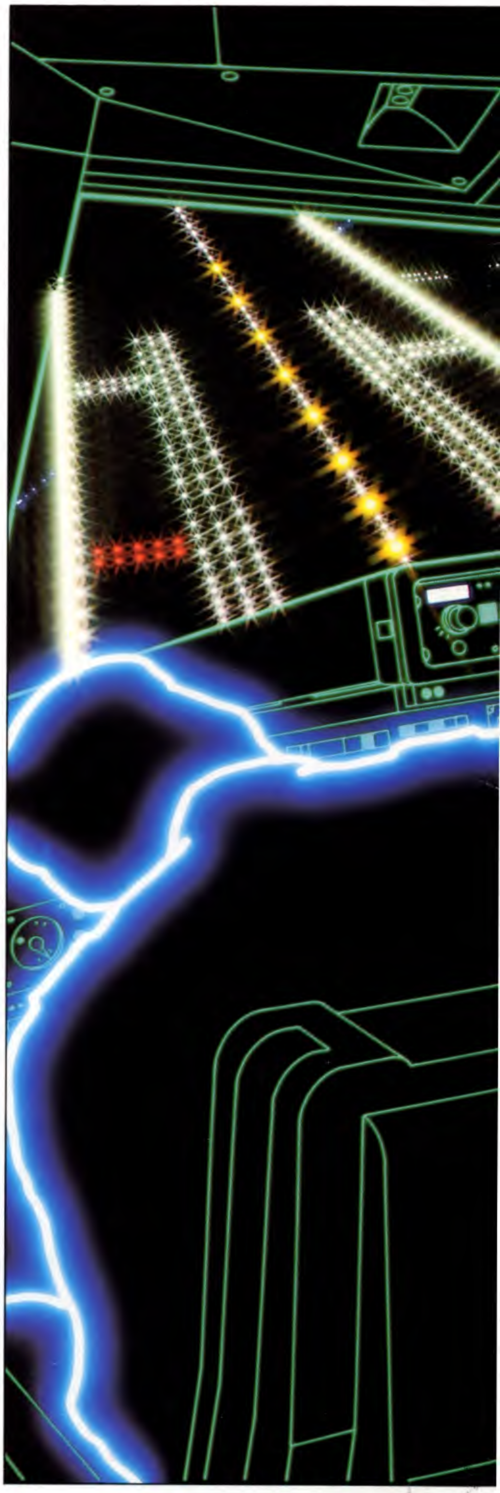
The third dial gives you an altitude reading. This has two hands, one for thousands, and the other for hundreds. The counter underneath calculates the drift of the aeroplane—as the runway is 100 metres wide, a drift of over +50 or −50 will cause you to miss it altogether.

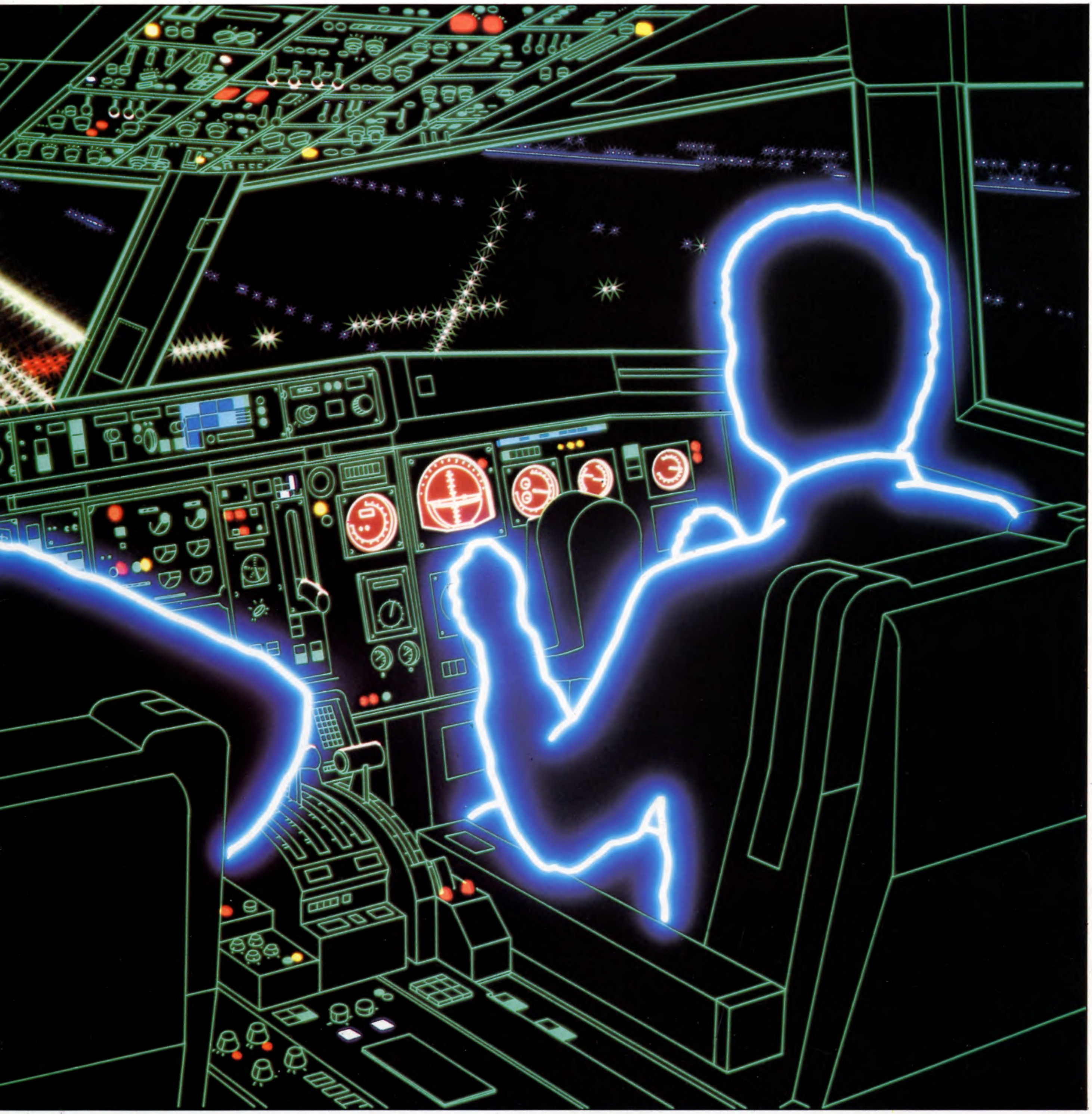
The last dial tells you the engine speed in revolutions per minute. The counter beneath lets you know the distance you are from the centre of the runway.

LANDING THE AEROPLANE

In the case of the Dragon and Tandy, the picture through the cockpit window becomes cleaner as you approach your destination—you can see the runway as you guide your aeroplane towards it. With the other computers you must centre the radar image of the runway.

When you assume the controls, weather conditions are fair, and the runway is due north. Landing like this is not hard, and the game would quickly lose its fun if you could not vary this. To add difficulty, you can specify the speed and direction of the wind: for instance, a howling gale from the side will make your job very much harder.





MOVING THE AEROPLANE

The range of controls you have closely approximates to the controls of the aeroplane—though you are pressing keys, rather than using a joystick.

In a real aeroplane, to control pitch—the up and down movement—the joystick is moved backwards or forwards, thereby moving the elevators on the tailplane upwards or downwards. You will be using two keys to create the same effect, and in the third section of the article you will enter the part of the program that does this.

The roll of the aeroplane—the side to side movement—is controlled by moving the joystick from side to side. This moves the ailerons—the control surfaces on the wings. Again you'll be using two keys to turn you either to the right or the left.

Your last two controls enable you to speed up or slow down the engine, essential for finely timing your landing, or making sure that you do not stall.

STALLING SPEED

Aeroplanes stall when they fall below a certain speed, which means that they literally drop from the sky. In this program, if your airspeed falls below 30 metres per second, the aeroplane will start to dive steeply, turning to one side as it plummets. If you've got enough height, quick action may save you, but a stall is dreaded by every pilot.

DIVIDING THE PROGRAM

The program is too long and complex to be given all at once, and so it has been split into three parts.

In this first part what you are doing is setting up the screen to show the interior of the cockpit, with its window, the four dials, which are labelled, and the labels for the counters.

The commands this involves will be familiar to most of you from other programs. Dragon and Tandy users, however, will come across a command that they may not have had to use before, PCOPY—unique to these computers—which will be explained in detail later.

In part two, the section of the program entered enables the dials and counters to become sensitive to the movement of the aeroplane, and a temporary command causes the aeroplane to fly randomly, without a pilot at the controls, so that you can watch the instrument panel functioning. The final section allows you to take control of the aeroplane, and assesses your landing technique so you can judge your progress.

DRAWING THE COCKPIT

To draw your cockpit, enter the first part of the program into your computer.

```
1 POKE 23658,8
11Ø GOTO 5ØØØ
5ØØØ LET PP=-1: LET RR=-1
5Ø1Ø LET C=PI/18Ø: LET PY=-2ØØØØ:
   LET PZ=2ØØØ: LET AS=15Ø
511Ø PLOT 1Ø,175: DRAW 235,Ø: DRAW
   Ø,-9Ø: DRAW -235,Ø: DRAW Ø,9Ø
512Ø FOR K=Ø TO 3: CIRCLE
   35+K*6Ø,5Ø,2Ø: NEXT K
513Ø PRINT AT 12,2;"SPEED□□□
   HORZN□□□ALT□□□□RPM"
515Ø PRINT AT 2Ø,Ø;"BEARING□□
   RUNWAY□□DRIFT□□DISTANCE"
517Ø PLOT 87,5Ø: DRAW 5,Ø: DRAW 3,-3:
   DRAW 3,3: DRAW 5,Ø
518Ø LET X=35: LET Y=5Ø: GOSUB 7ØØØ:
   LET X=155: GOSUB 7ØØØ: LET X=215:
   GOSUB 7ØØØ
69ØØ STOP
7ØØØ FOR K=Ø TO 2*PI STEP PI/5: PLOT
   X+17*SIN K,Y+17*COS K: DRAW 2*SIN
   K,2*COS K: NEXT K: RETURN
```

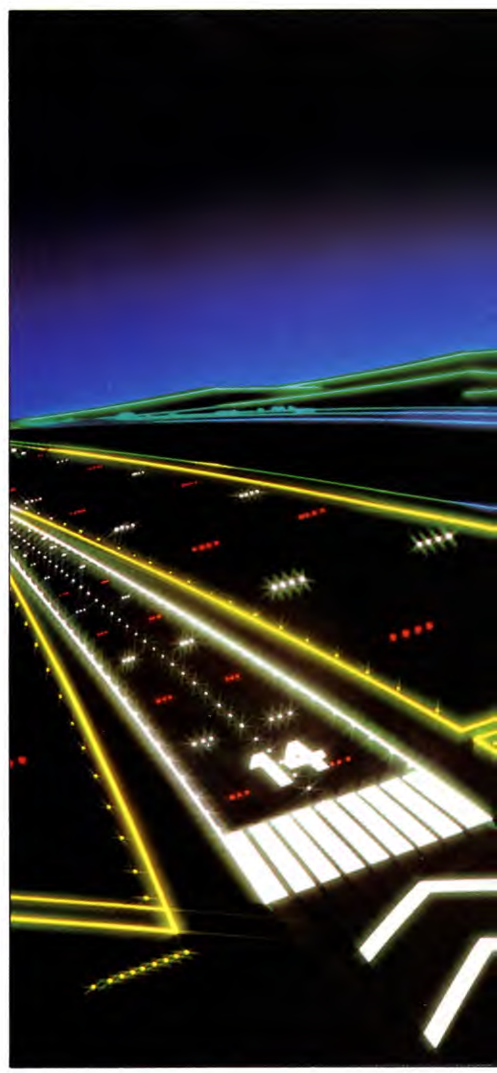
The POKE in Line 1 sets the computer to upper case mode. Lines 5ØØØ and 5Ø1Ø place the aeroplane in its position in the sky: 2000 metres up in the air, 20,000 metres from its destination, motionless; next week you will enter the lines that will make it move.

Line 511Ø draws the cockpit window, and the dials beneath are drawn by Line 512Ø, using a FOR . . . NEXT loop. The labels for the dials and counters are printed by Lines 513Ø and 515Ø. Line 517Ø draws a diagrammatic aeroplane in the horizon dial—the artificial horizon will not be drawn until the next part. Line 518Ø, and the GOSUB routine 7ØØØ, set the centres of the dials and draw the indicators round the three dials that need them: Airspeed, Altitude and RPM, using SIN and COS as explained on page 250.

If you RUN the program now, your simulated cockpit will appear on the screen.

To enable you to access the Commodore 64's high resolution graphics, this program is written in Simons' BASIC. This means that you cannot run it on the standard Commodore unless you fit a Simons' BASIC cartridge. But those of you who don't have a cartridge shouldn't despair. A machine code program that allows you to run these programs will be given soon in *INPUT*. Then you can enter and RUN any program written for Simons' BASIC, including this one. The only minor modification you will have to make is to preface each graphics command with a @.

```
51ØØ A$="AIRSPEED□HORIZON□
   ALTITUDE□□RPM"
511Ø B$="BEARING□RUNWAY□
   DRIFT□DISTANCE"
512Ø HIRES Ø,1:MULTI 4,Ø,5:
   COLOUR Ø,1
513Ø BLOCK Ø,11Ø,16Ø,2ØØ,2
514Ø TEXT Ø,12Ø,A$,3,1,5:
   TEXT Ø,175,B$,1,1,5
515Ø LINE Ø,171,16Ø,171,Ø:
   LINE Ø,2ØØ,16Ø,2ØØ,Ø
516Ø FOR Z=Ø TO 3:CIRCLE 2Ø+Z*4Ø,
   15Ø,15,15,Ø
517Ø IF Z=1 THEN TEXT 57,146,"⊟",
   Ø,1,1:NEXT Z
518Ø FOR K=Ø TO 9:PLOT (2Ø+Z*4Ø)+
   17*SIN(K*π/5),15Ø-19*COS
   (K*π/5),Ø:NEXT K,Z
```

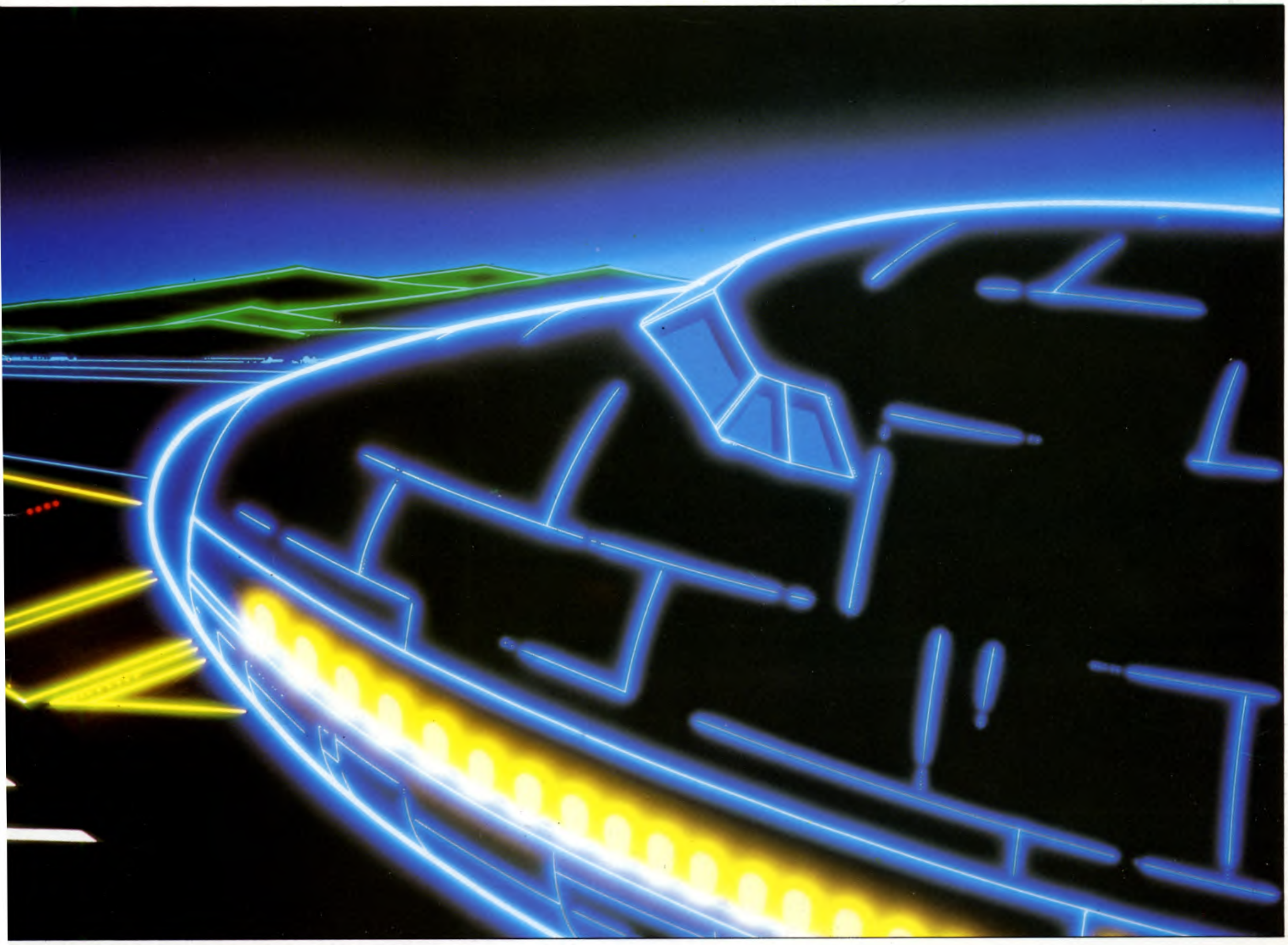


A$ and B$, as defined in Lines 51ØØ and 511Ø, contain the labels for the dials and counters. Line 513Ø draws a black block underneath the cockpit window, to contrast the dark interior of the aeroplane with the light blue window. Line 514Ø positions the labels, and Line 515Ø draws two lines to suggest that the labelled counters are set in a separate panel from the dials. Line 516Ø draws the circles for the dials (actually somewhat elliptical because of the way the graphics screen works), and the diagram of an aeroplane in the horizon dial. Line 518Ø draws the indicators around the other three dials.

So far, this program will show you a simulated flight deck. In the next part you will enter the part of the program that enables your aeroplane to fly. Line 519Ø must be deleted before adding this next section.

```
1Ø MODE1
15 PROCSCREEN: END
2Ø DEF PROCSCREEN
3Ø X=19Ø:GCOLØ,3
4Ø FOR P=Ø TO 3
5Ø MOVE X+1ØØ,25Ø
6Ø FOR T=Ø TO 2*PI+.3 STEP .3
7Ø DRAW X+1ØØ*COS(T),
   25Ø+1ØØ*SIN(T)
8Ø NEXT
9Ø X=X+3ØØ
1ØØ NEXT
11Ø X=17Ø
12Ø FOR P=Ø TO 3
13Ø X=19Ø+P*3ØØ
14Ø FOR T=Ø TO PI*2 STEP PI/5
15Ø MOVE X+1ØØ*SIN(T),
   25Ø+1ØØ*COS(T)
16Ø DRAW X+9Ø*SIN(T),
   25Ø+9Ø*COS(T)
17Ø NEXT
18Ø IF P=Ø THEN P=P+1
19Ø NEXT
2ØØ MOVE 42Ø,26Ø:DRAW47Ø,26Ø:
   DRAW49Ø,23Ø:DRAW51Ø,26Ø:
   DRAW56Ø,26Ø
21Ø MOVE 1ØØ,9ØØ:DRAW 118Ø,9ØØ:
   DRAW 118Ø,5ØØ:DRAW 1ØØ,5ØØ:
   DRAW 1ØØ,9ØØ
215 MOVE Ø,9Ø8:DRAW 1276,9Ø8:
   DRAW 1276,1Ø2Ø: DRAW Ø,1Ø2Ø:
   DRAW Ø,9Ø8
22Ø VDU 5:MOVE46,4ØØ:PRINT
   "AIR SPEED"
23Ø MOVE41Ø,4ØØ:PRINT"HORIZ"
24Ø MOVE742,4ØØ:PRINT"ALT"
25Ø MOVE1Ø42,4ØØ:PRINT"RPM"
26Ø MOVE78,1ØØ:PRINT"BEARING"
27Ø MOVE396,1ØØ:PRINT"RUNWAY"
28Ø MOVE71Ø,1ØØ:PRINT"DRIFT"
29Ø MOVE1Ø26,1ØØ:PRINT"DIST"
3ØØ VDU4:GCOL3,3:ENDPROC
```

Lines 3Ø to 1ØØ draw the control dials, using a FOR ... NEXT loop. Lines 11Ø to 19Ø draw the ten indicators round three of the dials—Altitude, Airspeed and RPM—the third dial, which displays the artificial horizon, does not need the indicators. The diagrammatic aeroplane, which appears in the horizon screen, is drawn by the Line 2ØØ. The artificial horizon







The instrument panel on the Commodore Flight Simulator ... and how it appears on the Acorn Flight Simulator

itself will not be drawn until the second part of the program is entered.

Line 21Ø draws the cockpit window, and the remaining lines label all the dials and counters.

RUNning the program at this stage will show you the interior of the cockpit, but the aeroplane can't fly, and neither can you control it yet.

```
10 PCLEAR8:PMODE4,1
20 DIM LE$(26)
30 FOR K=0 TO 26:READ LE$(K):NEXT
40 FOR K=0 TO 9:READ NU$(K):NEXT
50 DATA BR2,ND4R3D2NL3ND2BE2,
   ND4R3DGNL2FDNL3BU4BR2,
   NR3D4R3BU4BR2,ND4R2FD2GL2BE4BR,
   NR3D2NR2D2R3BU4BR2
60 DATA NR3D2NR2D2BE4BR,NR3D4R3U2
   LBE2BR,D4BR3U2NL3U2BR2,ND4BR2,
   BD4REU3L2R3BR2,D2ND2NF2E2BR2
70 DATA D4R3BU4BR2,ND4FREND4BR2,
   ND4F3DU4BR2,NR3D4R3U4BR2,
   ND4R3D2NL3BE2,NR3D4R3NHU4BR2
80 DATA ND4R3D2L2F2BU4BR2,BD4R3U2
   L3U2R3BR2,RND4RBR2,D4R2U4BR2,
   D3FEU3BR2,D4EFU4BR2
90 DATA DF2DBL2UE2UBR2,DFND2EUBR2,
   R3G3DR3BU4BR2
100 DATA NR2D4R2U4BR2,BDEND4BR2,
   R2D2L2D2R2BU4BR2,NR2BD2NR2BD2
   R2U4BR2,D2R2D2U4BR2,NR2D2R2
   D2L2BE4,D4R2U2L2BE2BR2,
   R2ND4BR2,NR2D4R2U2NL2U2BR2,
   NR2D2R2D2U4BR2
110 GOTO 5000
4000 FORK=1TOLEN(A$)
4010 B$=MID$(A$,K,1)
4020 IF B$>="0"ANDB$<="9" THEN
   DRAWNU$(VAL(B$)):GOTO4050
4030 IF B$="□" THENN=0 ELSEN=ASC
   (B$)-64
4040 DRAW LE$(N)
4050 NEXT:RETURN
5000 PP=-1:RR=-1
5010 PI=4*ATN(1):C=PI/180:
   PY=-20000:PZ=2000:AS=150
5110 PCLS:LINE(10,0)-(245,80),PSET,B
5120 FORK=0TO3:CIRCLE(35+K*60,
   120),25,5:NEXT
5130 DRAW"BM18,88S4":A$="AIRSPEED":
   GOSUB4000:DRAW"BM80,88":
   A$="HORIZON":GOSUB4000
5140 DRAW"BM140,88":A$="ALTITUDE":
   GOSUB4000:DRAW"BM208,88":
   A$="RPM":GOSUB4000
5150 DRAW"BM18,160":A$="BEARING":
   GOSUB4000:DRAW"BM82,152":
   A$="RUNWAY":GOSUB4000:
   DRAW"BM80,160":A$="BEARING":
   GOSUB4000
5160 DRAW"BM144,160":A$="DRIFT":
   GOSUB4000:DRAW"BM200,160":
   A$="DISTANCE":GOSUB4000
5170 DRAW"BM81,118R9F5E5R9"
5180 X=35:Y=120:GOSUB7000:X=155:
   GOSUB7000:X=215:GOSUB7000
5190 PCOPY3TO5:PCOPY3TO7:PCOPY4TO6:
   PCOPY4TO8:SCREEN1,1
5500 GOTO 5500
7000 FORK=0TO9:LINE(X+24*SIN(K*PI/5),
   Y-24*COS(K*PI/5))-(X+21*SIN
   (K*PI/5),Y-21*COS(K*PI/5)),PSET:
   NEXT:RETURN
```

The first part of this flight simulator program includes a command not so far dealt with in *INPUT*—PCOPY, which makes sure the screen image moves smoothly.

Lines 2Ø to 11Ø set up the arrays for writing on the graphics screen, as explained in detail on page 192, and this is carried out by the printing subroutine contained in Lines 4ØØØ to 4Ø5Ø.

Lines 5ØØØ and 5Ø1Ø position the aeroplane in the sky: 20,000 metres from the centre of the runway, at a level of 2000 metres, though at this stage it is motionless—it won't fly until you enter the lines in part two of the program.

The cockpit window is drawn by Line 511Ø. The next line, 512Ø, draws the circles for the dials, and Lines 513Ø to 516Ø label the dials. Line 517Ø draws a diagram of the aeroplane on the horizon dial. The indicators on the dials are drawn by Line 518Ø, and the subroutine in Line 7ØØØ.

The PCOPY commands in Line 519Ø draw the graphics onto unseen pages, and then copy them onto the screen. This aids background preservation—the background is renewed invisibly, and then transferred complete to the screen, so that there is no time lag when the background is updated. This means that when the aeroplane is flying and the dials move, they can all do so simultaneously—without PCOPY only one dial would move at a time.

RUN the program so far, and the cockpit will appear on your screen.

In the next part you will enter the commands that enable the aeroplane to fly, though, at that stage you will have no control over it. The aeroplane will be under the control of a manic autopilot, causing it to dive and climb randomly. The third part of the article will install the pilot's controls and you will finally have the passengers' lives in your hands.

UDGS MADE EASY

- DESIGNING THE CHARACTERS
- STORING UDGS IN A BANK
- **SAVE**ING UDGS ON TAPE OR DISK
- CONTROL KEYS



If you're fed up with laboriously working out UDGs on paper, then type in this program and take a rest while the machine does the work and you plot the shape you want

UDGs are one of the most versatile tools available to the graphics programmer—and they have been exploited in numerous programs in *INPUT*. But one of the drawbacks, if you want to design your own, is the effort that goes into first plotting them on paper, then working out the DATA, and finally POKEing into memory so you can display them on screen. And if you've got it wrong, it takes a long time to see the results.

So here's a program to change all that. It replaces the piece of squared paper with a neat screen display which lets you plot in pixels where you want them. Instead of you having to work out the DATA values by hand, it tots them up automatically.

But the real strength of the program is that it also lets you see the UDG as it builds up—life size—and you don't have to bother about POKEing the DATA into another program. Various control options let you adapt the UDG at the touch of a button—perhaps you would like to see the inverse, or turn it round from left to right.

When you've created the UDG you're looking for—and you have nearly 20 million million million options—you will want to be able to keep it. So the program lets you store a bank of completed UDGs that you can call back when you want them. You can SAVE the UDGs out of the program and onto tape, and you can then LOAD them back into any program for use on screen—or back into the UDG generator for further editing.

The program is in two parts. The first part of the listing, which follows, allows you to create the basic UDG generator. It sets up a grid on screen, and allows you to move around the display to plot the pattern of pixels you want in the finished UDG. In the following part of the article, you will add the routines which enable you to modify the UDG in more subtle ways, and to display it.

You can save having to type it in again next time by SAVEing it on tape when you have entered it.

When you type in the program and RUN it, you will see a square 8 × 8 grid in the middle of the screen. This is the 'graph paper' on which you are going to plan your UDG. So you can select which squares to block in, this sets a pixel.

You have a cursor, which takes the form of a white square. This square is not printed in a special colour, but is the BRIGHT version of whatever it overprints. This is so you can see it, whether or not the background square has been set. The program is written so that you move the cursor with the cursor keys and press Ø to set, or reset, the pixel.

If you want to change these keys to a more personally convenient set, simply change the characters inside the IF I$ = . . . statements in Lines 652Ø to 6555 to those of the keys you wish to use.

You can use this method to set the program up for use with some joystick interfaces, but if you have a Kempston-compatible interface, you will need to change these lines even more. Instead of the INKEY$ in Line 651Ø, you need to use IN, as pages 464 to 469 explain.

Whichever keys you decide to use, you should be careful not to use the same keys as the control keys. This part of the program only uses three control keys (these are ex-

plained further on in the article) but the next part will use more. Altogether, C, I, M, P, R, S and T are reserved.

Using either the keyboard or a joystick, the cursor can move around inside the grid in the middle of the screen. If you press 'fire', key Ø on the keyboard, then you will see the colour of the cursor change to indicate that the pixel 'underneath' the cursor is now on. If you move the cursor away from that position, you will see that the square is now shaded in. Carry on in this way to build up your design.

The program has a number of options which you can access at any time while editing a UDG. The first of these is to store the UDG. To get this option, simply press key S—this is the first of the control keys mentioned earlier.

You will then be asked what position in the UDG bank you wish your present design to occupy. The UDG bank is represented by the letters A to U, displayed above the grid. When you have entered a letter between A and U to make your choice, there will be a slight delay while the computer POKEs your UDG into place, after which you will see it appear in the bank above the grid. You are then able to continue.

The next option is to pick up a UDG. This lets you call up any of the bank of UDGs for editing. When you choose this option, by pressing the P key, you will be asked which UDG you want to pick up. After pressing the appropriate key, the UDGs you want will appear in the grid.

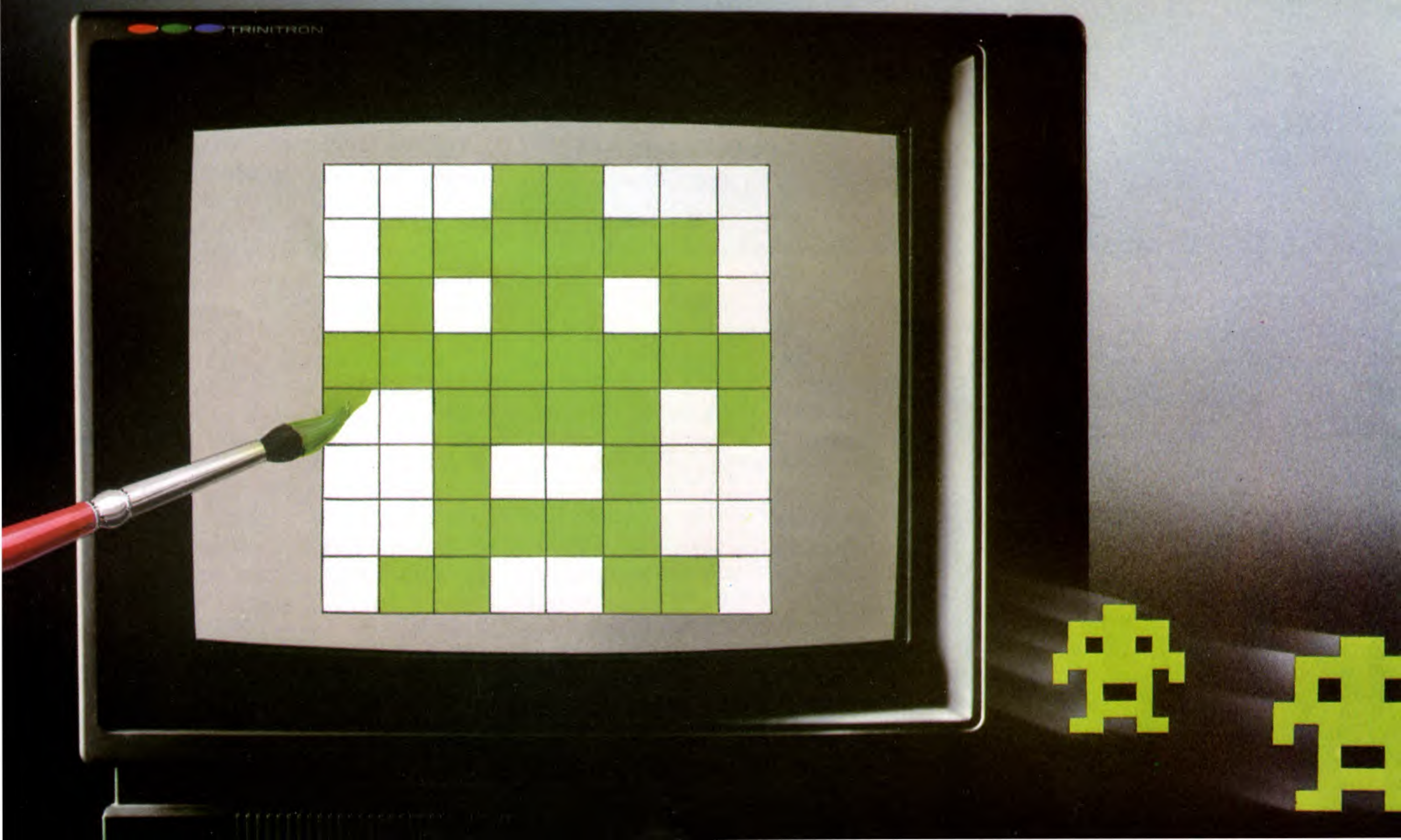
SAVEING UDGs ON TAPE

Although the UDG editor is fine as it is, you need to be able to SAVE your finished designs to tape for use in your own programs. To match this, there is a LOAD option so you can LOAD previously SAVEd UDGs for re-editing.

You can use either the SAVE or the LOAD facility by pressing the T key. You will be asked whether you wish to SAVE or LOAD, and then the computer will do this. When you SAVE to tape, the bytes of the 21 UDGs currently stored are SAVEd as a block of memory—so don't forget to store the UDG you are working on before SAVEing the bank.

The final part of this article will give you more information about how you can use these facilities to design and use lots of UDGs and includes detailed instructions on how to call them into another program. It also adds a number of extra useful features.

```
5 CLEAR USR "A"-16
6 POKE 23675,PEEK 23675-16
8 BORDER 4: PAPER 7: INK Ø: CLS
10 FOR N=USR "A" TO USR "B"+7:
   READ A: POKE N,A: NEXT N
15 POKE 23675,PEEK 23675+16
20 POKE 23658,8
30 LET X=11: LET Y=8: LET NX=X: LET
   NY=Y
100 LET A$="": FOR N=144 TO 164: LET
   A$=A$+CHR$ N: NEXT N
110 LET B$="": FOR N=65 TO 85: LET
   B$=B$+CHR$ N: NEXT N
1000 FOR N=87 TO 151 STEP 8: PLOT N,48:
   DRAW Ø,64: NEXT N
1010 FOR N=48 TO 112 STEP 8: PLOT 87,N:
   DRAW 64,Ø: NEXT N
2000 PRINT AT 3,5;A$
2010 PRINT INVERSE 1;AT 2,5;B$
2400 GOSUB 6500
2500 IF INKEY$="P" THEN GOTO 5000
2510 IF INKEY$="S" THEN GOTO 5100
2520 IF INKEY$="T" THEN GOTO 5200
3900 GOTO 2000
5000 INPUT "WHICH CHARACTER (A-U)?",
   LINE C$
5010 IF C$<CHR$ 65 OR C$>CHR$ 85
   THEN GOTO 5000
5020 LET D=CODE C$+79: LET C$=CHR$
   D: GOSUB 6000: GOTO 2000
5100 INPUT "STORE IN WHICH CHARACTER
   (A-U)?", LINE C$
5110 IF C$<CHR$ 65 OR C$>CHR$ 85
   THEN GOTO 5100
5120 PRINT AT 18,10;"STORING NOW"
5130 FOR N=USR C$ TO USR C$+7
5140 LET R=Ø: LET BIT=128: FOR M=Ø
   TO 7
5150 IF PEEK (18432+(N-USR
```



```
   C$)*32+M+11)<>1 THEN LET
   R=R+BIT
5160 LET BIT=BIT/2: NEXT M: POKE N,R
5170 NEXT N: PRINT AT 18,0;"□"; TAB
   31;"□": GOTO 1000
5200 INPUT "L(OAD) OR S(AVE)?", LINE C$:
   IF C$<>"L" AND C$<>"S" THEN
   GOTO 1000
5220 IF C$="L" THEN GOTO 5250
5230 INPUT "ENTER FILENAME", LINE N$: IF
   N$="" OR LEN N$>10 THEN GOTO
   5230
5240 SAVE N$CODE USR "A",168: GOTO 100
5250 INPUT "ENTER FILENAME", LINE N$: IF
   LEN N$>10 THEN GOTO 5250
5260 PRINT AT 19,0;: LOAD N$CODE USR
   "A": PRINT AT 20,0;"□";TAB 31;"□":
   GOTO 100
6000 LET B=USR "A"+8*(CODE C$-144)
6010 POKE 23675,PEEK 23675-16
6020 FOR N=0 TO 7
6030 LET V=PEEK (B+N)
6040 LET BIT=128: FOR M=0 TO 7
6050 IF V>=BIT THEN PRINT AT
   8+N,11+M;CHR$ 144: LET V=V-BIT:
   GOTO 6060
6055 PRINT AT 8+N,11+M;CHR$ 145
6060 LET BIT=BIT/2: NEXT M
6070 NEXT N: POKE 23675,PEEK 23675+16:
   RETURN
6500 POKE 22528+32*Y+X,120: PAUSE 0
6510 LET I$=INKEY$
6520 IF I$="5" AND X>11 THEN LET
   NX=X-1
6530 IF I$="8" AND X<18 THEN LET
   NX=X+1
6540 IF I$="6" AND Y<15 THEN LET
   NY=Y+1
6550 IF I$="7" AND Y>8 THEN LET
   NY=Y-1
6552 IF I$="0" THEN GOSUB 7000
6555 IF I$="" THEN GOTO 6580
6560 POKE 22528+Y*32+X,56
6570 LET X=NX: LET Y=NY
6580 POKE 22528+Y*32+X,120
6590 RETURN
7000 POKE 23675,PEEK 23675-16
7010 IF PEEK (18432+(Y-8)*32+X)=1
   THEN PRINT BRIGHT 1;AT Y,X;CHR$ 144:
   GOTO 7030
7020 PRINT BRIGHT 1;AT Y,X;CHR$ 145
7030 POKE 23675,PEEK 23675+16: RETURN
9000 DATA 85, 171, 85, 171,85, 171,85,
   255,1,1,1,1,1,1,1,255
```

C= C=

The Commodore and Vic programs both use the same control keys and work in the same way. The Vic program is for either an unexpanded Vic or a machine with 3K expansion. It will not work on 8 or 16K expanded Vics—but you can take out the expansion pack, of course.

When you type in and RUN the Commodore programs, there is a delay of about a minute while the computer clears the space for the new characters and POKEs the existing set into RAM.

As soon as this has been done, the screen display changes to give you the grid in which you can define your characters.

You can move the cursor around inside the grid using these keys: z is left; x is right; ; is up; and / is down. When you are over the pixel position you want to set, press the SHIFT key. You can fill in a lot of pixels quickly by pressing SHIFT LOCK on, and moving at the same time.

As well as being able to set pixels, you can also turn them off again by pressing the Commodore key.

By moving around inside this grid, setting the pixels you want, you can create your UDGs very quickly. But the program helps you even more by providing a set of extra facilities. The first of these is the twin line of characters below the grid. These display the present bank of UDGs (you can have up to 26) and their positions. The top line is simply the letters A to Z, while underneath this is a line of UDGs that starts off as the character set but can be replaced with your own, new UDGs. This storage bank is used when it comes to storing your characters.

HOW TO STORE A UDG

When you are satisfied with the UDG design, first press the left-arrow key. The computer will then ask you under which letter you want your UDG stored. Check the display of the present bank to see which character you want to replace with your newly defined UDG.

When you have decided where you want to put it, press the key for that letter. After a very brief pause, your character will take its place in the bank at the bottom of the screen.

Once you have designed and stored a character in the bank, you can retrieve it onto the editing grid at any time by pressing the @ key, again followed by the character you want.

There is one more control character in this half of the program—the £ key. This gives you access to the tape mode, so that you can either SAVE your present bank of characters, or LOAD in a previously SAVEd one.

After you have pressed this key, the computer asks you whether you want to SAVE or LOAD. It assumes you have the tape already in the right place and ready to start LOADing/SAVEing. When the computer has finished SAVEing, or LOADing the character data, the computer returns you to the editing screen to continue.

The next article in this series will complete the program, and will use several more control keys to rotate, mirror, invert, clear the grid, and print the DATA to a printer. There will also be instructions on calling up the UDGs for use within another program.

C=

```
10 POKE 51,255:POKE 52,47:POKE
   55,255:POKE 56,47:CLR
15 POKE 56334,0:POKE 1,51
20 FOR Z=0 TO 1023:POKE 12288+Z,
   PEEK(53248+Z):POKE 13312+Z,
   PEEK(53248+Z):NEXT Z
25 POKE 1, 55:POKE 56334,1
30 PRINT "□■";CHR$(8):POKE 53280,1:
   POKE 53281,1:POKE 53272,28
```



```
34 PRINT TAB(10);"■CHARACTER□□
   GENERATOR"
38 PRINT TAB(10);"■□□□□□
   □□□□□□□□□□□□□□
   □□"
40 A$="ABCDEFGHIJKLMNOPQRSTUVW
   XYZ"
50 X=0:Y=0:Z(0)=128:Z(1)=64:
   Z(2)=32:Z(3)=16:Z(4)=8:
   Z(5)=4:Z(6)=2:Z(7)=1
100 PRINT "■■■■■■■■
   ■■";TAB(12);"12345678"
110 FOR Z=1 TO 8:PRINT TAB(10);Z;
   "■■□□□□□□□□   ":
   NEXT
120 PRINT TAB(12);"■□□□
   □□□□□□":PRINT "■";
   TAB(7);A$:PRINT TAB(7);
122 FOR Z=1 TO 26:PRINT"■-";:
   NEXT Z:PRINT:PRINT TAB(7);A$
130 XX=X:YY=Y:C=1396+X+Y*40:
   CC=PEEK(C)
140 POKE C,233:P=PEEK(197):POKE 198,0
150 IF P=12 AND X>0 THEN X=X-1
155 IF P=23 AND X<7 THEN X=X+1
160 IF P=50 AND Y>0 THEN Y=Y-1
165 IF P=55 AND Y<7 THEN Y=Y+1
170 IF PEEK(653)=1 THEN POKE C,230:
   GOTO 130
180 IF PEEK(653)=2 THEN POKE C,207:
   GOTO 130
200 POKE C,CC
250 IF P=57 THEN 1200
280 IF P=46 THEN 1600
290 IF P=48 THEN 1400
300 GOTO 130
1200 GOSUB 1500
1235 FOR Z=0 TO 7:T=0:FOR ZZ=0 TO7
1240 C1=1396+ZZ+Z*40:C2=PEEK(C1)
1250 IF C2=230 THEN T=T+Z(ZZ)
1260 NEXT ZZ:POKE 12288+(A*8)+Z,T:
   POKE 12288+Z,T
1280 NEXT Z:GOTO 130
1400 POKE 198,0:POKE 53272,21:N$="":
   INPUT "♥ENTER NAME";N$:IF N$=""
   THEN 30
1405 PRINT "■(S)AVE OR (L)OAD?":OU=0
1406 GET K$:IF K$="S" THEN
   OU=1:GOTO 1410
1407 IF K$<>"L" THEN 1406
1410 PRINT "♥":R$=CHR$(13):OPEN
   1,1,OU,N$:REM DISK:- OPEN
   1,8,OU,"@:"+N$
1413 FOR Z=12296 TO 12504:IF OU=1
   THEN Z$=STR$(PEEK(Z)):Z$=
   RIGHT$(Z$,LEN(Z$)-1)
1414 IF OU=1 THEN PRINT#1,Z$R$:
   GOTO1420
1415 INPUT#1,Z$:POKE Z,VAL(Z$)
1420 NEXT Z:CLOSE1:GOTO 30
1500 PRINT "■■■■■";TAB(14);
   "ENTER LETTER":FOR Z=1 TO 25:NEXT Z
1505 PRINT "■■■■■";TAB(14);
   "□□□□□□□□□□□□□□"
1510 GET A$:IF A$<"A" OR A$>"Z"
   THEN 1500
1520 A=ASC(A$)-64:RETURN
1600 GOSUB 1500
1610 FOR ZZ=0 TO7
1611 AA=PEEK(12288+A*8+ZZ)
1612 FOR Z1=0 TO 7
1614 IF AA-Z(Z1)=>0 THEN AA=
   AA-Z(Z1):POKE 1396+Z1+ZZ*40,
   230:GOTO 1620
1616 POKE 1396+Z1+ZZ*40,207
1620 NEXT Z1,ZZ:GOTO 130
```

```
10 PRINT CHR$(8):POKE 51,255:
   POKE 52,27:POKE 55,255:POKE 56,27:CLR
20 FOR Z=0 TO 511:POKE 7168+Z,
   PEEK(32768+Z):NEXT Z
30 PRINT "♥":POKE 36879,25:POKE
   36869,255
33 FOR Z=0 TO 7:POKE 7168+Z,255:NEXT Z
35 PRINT "■■CHARACTER□□
   GENERATOR"
38 PRINT "■■□□□□□□□□
   □□□□□□□□□□□□"
40 A$="ABCDEFGHIJKLMNOPQRSTUV"
50 X=0:Y=0:Z(0)=128:Z(1)=64:
   Z(2)=32:Z(3)=16:Z(4)=8:
   Z(5)=4:Z(6)=2:Z(7)=1
```

```
100 PRINT "■■■■■■■■■■
    □□12345678"
110 FOR Z=1 TO8:PRINT Z;"▮▮■
    □□□□□□□□□□□":NEXT
120 PRINT
    "■□□□□□□□□□□□□
    ":PRINT "■"A$:PRINT A$
130 XX=X:YY=Y:C=7880+X+Y*22:
    CC=PEEK(C)
140 POKE C,233:P=PEEK(197):POKE 198,0
150 IF P=33 AND X>0 THEN X=X-1
155 IF P=26 AND X<7 THEN X=X+1
160 IF P=22 AND Y>0 THEN Y=Y-1
165 IF P=30 AND Y<7 THEN Y=Y+1
170 IF PEEK(653)=1 THEN POKE C,230:
    GOTO 130
180 IF PEEK(653)=2 THEN POKE C,207:
    GOTO 130
200 POKE C,CC
250 IF P=8 THEN 1200
280 IF P=53 THEN 1600
290 IF P=6 THEN 1400
300 GOTO 130
1200 GOSUB 1500
1235 FOR Z=0 TO 7:T=0:FOR
     ZZ=0 TO7
1240 C1=7880+ZZ+Z*22:C2=PEEK(C1)
1250 IF C2=230 THEN T=T+Z(ZZ)
1260 NEXT ZZ:POKE 7168+(A*8)+Z,T:
     POKE 7168+Z,T
1280 NEXT Z:GOTO 130
1400 POKE 198,0:POKE 36869,240:
     NS="":INPUT "◘ENTER NAME";
     N$:IF N$="" THEN 30
1405 PRINT "■(S)AVE OR (L)OAD?":
     OU=0
1406 GET K$:IF K$="S" THEN OU=1:
     GOTO 1410
1407 IF K$<>"L" THEN 1406
1410 PRINT "◘":OPEN 1,1,OU,N$:REM
     DISK:- OPEN 1,8,OU,"@:"+N$
1413 FOR Z=7176 TO 7352:IF OU=1 THEN
     PRINT#1,PEEK(Z):GOTO 1420
1415 INPUT#1,ZZ:POKE Z,ZZ
1420 NEXT Z:CLOSE1:GOTO 30
1500 PRINT "■■■■■□□□□□
     ENTER LETTER":FOR Z=1 TO 25:NEXT Z
1505 PRINT "■■■■■□□□□□
     □□□□□□□□□□□□□□"
1510 GET A$:IF A$<"A" OR A$>"V"
     THEN 1500
1520 A=ASC(A$)-64:RETURN
1600 GOSUB 1500
1610 FOR ZZ=0 TO7
1611 AA=PEEK(7168+A*8+ZZ)
1612 FOR Z1=0 TO 7
1614 IF AA-Z(Z1)=>0 THEN AA=
     AA-Z(Z1):POKE 7880+Z1+ZZ*22,
     230:GOTO 1620
1616 POKE 7880+Z1+ZZ*22,207
1620 NEXT Z1,ZZ:GOTO 130
```

When you type in and RUN the Acorn program, the computer draws a blank grid and a line of characters is PRINTed above it.

The grid is the main part of the UDG generator and represents the 8 × 8 group of pixels in the UDG. You can move the cursor around inside it, setting pixels where you want, to build up your character. You can tell when a pixel is set, since a filled-in circle is printed in the grid square to represent a filled-in pixel—but to start with, the grid is blank. You can move around with a variety of keys, depending upon what you want to do.

To move and leave the pixels unchanged, you can use these keys: P is up; L is down; Z is left; and X is right. RETURN also leaves the pixels unchanged, and at the same time brings you to the start of the next line.

If you press the DELETE key, the pixel beneath the cursor is set to 0 (or deleted, if it had previously been set) and the cursor is then moved one space to the left. To delete a pixel and move the cursor to the right, you can press the space bar.

The full stop key sets a pixel, and moves the cursor to the right, while the comma key sets a pixel and moves the cursor to the left.

Using a combination of these keys, you can quickly and easily move around the grid, setting various pixels to create your character. As you build up your UDG, you can see its actual size from the character above the grid. This is updated whenever you move.

THE UDG STORE

Once you have finished your design, you can store it in memory in a bank of 32 UDGs. The current bank (part of the character set) is displayed at the top of the screen. It's also possible to call a UDG back from this store for further editing, if you wish.

The up-arrow beneath one of the characters indicates which position in the bank is currently available. If you store a UDG, the computer puts it in the position pointed to by the arrow, and if you take a character from the bank on to the grid for further editing, you will get the one above the arrow.

You need to be able to move the arrow to specify which character you want to use. Press the semi-colon key to move it to the right. To move left, press the minus-sign key.

When you have positioned the arrow, to store a character in the bank, press CONTROL and A. To get a character back into the grid again from the bank, press CONTROL and G. (A stands for Assign, and G for Get—it is often easier to remember the names, rather than just the letters.)

After you have finished a whole bank, or as much of a bank as you want to use, you can SAVE the bank to tape. This facility means that you have a permanent record of the UDGs which you can LOAD back into the computer at any time. The next part of this article will explain how you call up the UDG for use in your own programs. For now, you can LOAD it back into the program, for later editing.

You can use the SAVE facility by pressing CONTROL and S together. To LOAD a previously SAVEd bank back into memory from the program, press CONTROL and L together.

This half of the program has just one more control key available—CONTROL and E. This ends the program, and returns you to BASIC. But next time, you'll see how to add extra facilities, including a print-out routine.

If you use a disk drive, omit Line 20.

```
10 *FX11,0
15 *FX4,1
20 *OPT2,1
25 *OPT1,1
30 MODE1:DIM A(31),B(31),C(7)
40 VDU 23,95,0,0,0,0,0,0,0,255:
   FOR T=0 TO 31:B(T)=T?&C00:
   NEXT:*FX20,0
50 X=0:Y=0:PT=0:PX=508:PY=252:
   DX=260:DY=260
60 PROCASS:VDU23;8202;0;0;0;
70 FOR T=0 TO 23:READ T?&C00:NEXT
80 CLS:PROCDISPLAY:REPEAT :
   PROCINST:UNTIL A$=CHR$(5)
90 CLS:INPUT""ARE YOU SURE (Y/N)";A$
100 IF A$<>"Y" THEN 80 ELSE END
110 DEF PROCGRID
120 GCOL0,1
130 FOR P=0 TO 3:MOVE PX-DX*(P=3),
PY-DY*(P=1):DRAW PX-DX*
   (P<>2),PY-DY*(P<>0)
140 MOVE 88-1040*(P=3),888-48*
   (P=1):DRAW 88-1040*(P<>2),
   888-48*(P<>0):NEXT
150 GCOL3,3:ENDPROC
160 DEF PROCINST
170 A$=INKEY$(1):IF A$="" THEN ENDPROC
180 IF A$=";" THEN PROCPOINT(1):
    ENDPROC
190 IF A$="-" THEN PROCPOINT(-1):
    ENDPROC
200 IF A$=CHR$(7) THEN PROCGET:
    ENDPROC
210 IF A$=CHR$(1) THEN PROCASIGN:
    ENDPROC
220 IF A$=CHR$(12) THEN PROCLOAD:
    ENDPROC
230 IF A$=CHR$(19) THEN PROCSAVE:
    ENDPROC
290 IF A$="," THEN PROCIN:X=X-1:
    IF X=-1 THEN Y=Y-1
```

```
300 IF A$="." THEN PROCIN:X=X+1:
    IF X=8 THEN Y=Y+1
310 IF A$="□" THEN PROCDEL:X=X+1:
    IF X=8 THEN Y=Y+1
320 IF A$=CHR$(13) THEN X=Ø:Y=Y+1
330 IF A$=CHR$(127) THEN PROCDEL:
    X=X-1:IF X=-1 THEN Y=Y-1
350 X=(X-(A$="X")+(A$="Z")) AND
    7: Y=(Y+(A$="P")-(A$="L"))
    AND 7
360 CALL MC:VDU5:MOVE(16+X)*32,
    1Ø23-(16+Y)*32:VDU224,4
370 ENDPROC
380 DEF PROCDEL:?(&C18+Y)=
    ?(&C18+Y) AND (255-2∧(7-X)):
    ENDPROC
390 DEF PROCIN:?(&C18+Y)=
    ?(&C18+Y) OR 2∧(7-X):ENDPROC
4ØØ DEF PROCNOS
41Ø CALL MC:VDU5:MOVE(16+X)*32,
    1Ø23-(16+Y)*32:VDU224,4:
    ENDPROC
42Ø DEF PROCDISPLAY
43Ø PROCMOVEUDG(1):VDU31,3,3:FOR
    T=224 TO 255:VDU T:NEXT:
    PROCMOVEUDG(Ø)
44Ø VDU5:MOVE(16+X)*32,1Ø23-
    (16+Y)*32:VDU224,4:PROCPOINT(Ø):
    PROCGRID:PROCNOS:ENDPROC
45Ø DEF PROCMOVEUDG(F)
46Ø FOR T=Ø TO 31: IF F=Ø THEN 48Ø
47Ø A(T)=T?&CØØ:T?&CØØ=B(T):
    GOTO 49Ø
48Ø B(T)=T?&CØØ:T?&CØØ=A(T)
49Ø NEXT:ENDPROC
5ØØ DEF PROCLOAD
51Ø VDU 4: CLS:PROCMOVEUDG(1)
52Ø INPUT"ARE YOU SURE (Y/N) ",A$:
    IF A$<>"Y" THEN 55Ø
53Ø INPUT"FILENAME",A$:IF LENA$>8
    THEN 53Ø
54Ø H=OPENIN(A$):FOR T=&CØØ TO
    &CFF:?T=BGET#H:NEXT:CLOSE#H
55Ø PROCMOVEUDG(Ø):CLS:PROCDISPLAY:
    ENDPROC
56Ø DEF PROCSAVE
57Ø VDU 4:CLS:PROCMOVEUDG(1)
58Ø INPUT"ARE YOU SURE (Y/N)□";A$:
    IF A$<>"Y" THEN 61Ø
59Ø INPUT"FILENAME";A$:IF LENA$>8 OR
    A$="" THEN 59Ø
6ØØ H=OPENOUT(A$):FOR T=&CØØ TO
    &CFF:BPUT#H,?T:NEXT:CLOSE#H
61Ø PROCMOVEUDG(Ø):CLS:PROCDISPLAY:
    ENDPROC
62Ø DEF PROCPOINT(F)
63Ø PRINTTAB(3+PT,5)"□":PT=(PT+F)
    AND 31:PRINTTAB(3+PT,5)"∧":
    ENDPROC
64Ø DEF PROCASIGN
65Ø PROCMOVEUDG(1):FOR T=Ø TO 7:
    T?(&CØØ+PT*8)=A(T+24):NEXT
66Ø PROCMOVEUDG(Ø):PROCDISPLAY:
    ENDPROC
67Ø DEF PROCGET
68Ø PROCMOVEUDG(1):FOR T=Ø TO 7:
    A(T+24)=T?(&CØØ+PT*8):NEXT:
    PROCMOVEUDG(Ø)
69Ø FOR T=Ø TO 7:T?(&CØØ+24)=
    A(T+24):NEXT:PROCDISPLAY:
    ENDPROC
77Ø DATA 255,255,255,255,255,255,
    255,255,Ø,Ø,Ø,24,24,Ø,Ø,Ø,Ø,
    6Ø,126,126,126,126,6Ø,Ø
78Ø DEF PROCASS
79Ø DIM MC 2ØØ:FOR T=Ø TO 2 STEP 2:
    P%=MC
8ØØ [OPT T
81Ø LDA #24:STA &7Ø:LDA #12:STA &71:
    LDY #7
82Ø .M3: JSR PSN:LDA (&7Ø),Y:LDX
    #8:.M4:ASL A:PHA:LDA #225:ADC #Ø
83Ø .M5: JSR &FFEE:PLA:DEX:BNE
    M4:DEY:BPL M3:LDA #31:JSR
    &FFEE:LDA #19:JSR &FFEE:LDA
    #13:JSR &FFEE:LDA #227:JMP &FFEE
84Ø .PSN: PHA:LDA #31:JSR &FFEE:LDA
    #16:JSR &FFEE:TYA:CLC:ADC #16:JSR
    &FFEE:PLA:RTS
85Ø .ROT:LDY #Ø:.L2: LDX #7:LDA
    &C18,Y:.L3:LSR A:ROR &7Ø,X:DEX:BPL
    L3:INY:CPY #8:BNE L2:.L4 LDA
    &6F,Y:STA &C17,Y:DEY:BNE L4:RTS
86Ø .FIN:]:NEXT
87Ø IF FIN<>MC+1Ø1 THEN PRINT
    "MACHINE CODE WRONG, CHECK VERY
    CAREFULLY":END
88Ø ENDPROC
```

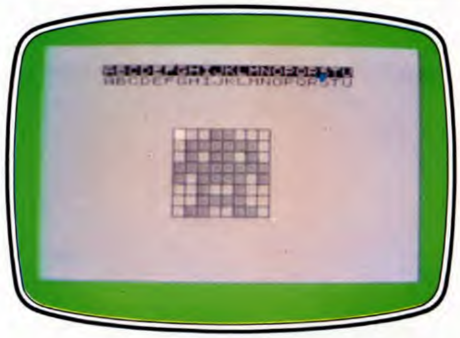
T

When you type in and RUN the program, you will be presented with two questions, which determine the sort of UDG you define.

The first of these asks which PMODE you want to use. PMODE4 gives you high resolution UDGs, but in only either green and black or buff and black. The second, PMODE3, gives you half as many pixels across (the pixels are twice as wide) but allows you to have four colours.

The second of the INPUTs lets you determine which colour sets you want. Screen Ø in PMODE4 gives green and black, while in PMODE3 it gives red, blue, green, and yellow. Screen 1 gives you buff and black in PMODE4, and cyan, magenta, orange and buff in PMODE3.

After you have made these two choices, you cannot then change the PMODE without reRUNning the program but you *can* change the screen.



The Spectrum lets you make UDGs ...

The computer then presents you with the editing grid which forms the basis for the UDG design. You can move a cursor around inside this grid using the cursor keys and, by moving to the pixel you want, you can change it, by turning it on or off.

With the cursor positioned over the pixel, press the ENTER key to set the pixel. You delete pixels by plotting them in a different colour, as explained later in this article.

Using these keys, you can design your UDGs quickly and easily. Next week the second half of the program includes, among other things, an option for joystick.

As you can see, the grid is not the only square on the screen: there are eight smaller ones beneath the main grid, and two smaller ones on the right of the grid.

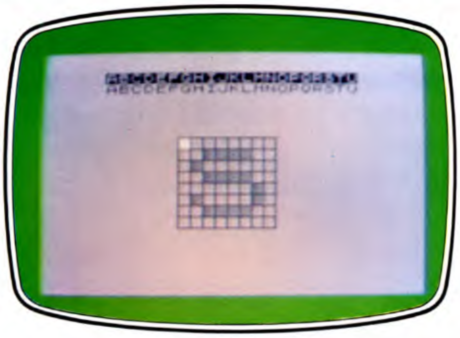
The two on the right of the grid are actual-size versions of the UDG you are editing both normal and reversed. As the pixels are changed, these two 'models' are updated, enabling constant monitoring of your design.

The eight squares below the main 'drawing board' are for storing the current bank of UDGs, or what you have already defined. This means that you can define up to eight UDGs and keep them all in memory at any time. Some of these are not always visible. This happens when the colours of the plotted pixels are the same as the background colour. These are not updated all the time, but only when you store the newly edited UDG, as explained later in this article.

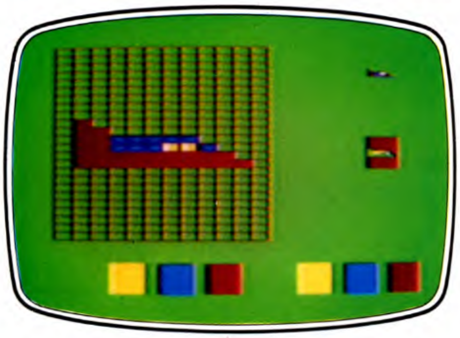
CONTROL KEYS

This program has a number of control keys, to enable you to use its numerous options at the press of a key.

Two of these control keys go hand in hand, and are fundamental. These are S and G. S stores the UDG currently in the grid in the bank. After pressing the S key, the computer waits for you to press another key—a number between 1 and 8. This number determines the position of the UDG in the bank. This



or use the computer's own characters



The Dragon allows larger UDGs

enables you to replace any of the bank with updated versions.

G, again followed by a number between 1 and 8, does the opposite—it takes the UDG out of the bank and places it in the grid. Although the program uses a machine code routine, the process takes a few seconds.

Picking up a character like this does not change the contents of the relevant bank position—you could pick up the same UDG as many times as you like, as long as you do not store a new design in its place.

This part of the program has two more control keys. If you press C, you can change the colour. If you are in the four-colour mode, you can change between any of the four colours available. If in the two-colour mode you can switch between either of the two colours. You might like to do this to rub out any of the pixels you have set by mistake. After pressing the C key, tell the computer the number of the new colour you want.

The last control key in this half of the program is T. This allows you to either LOAD in a previously SAVEd bank of UDGs, or to SAVE the present one.

T

If you have a Tandy, you should make these changes to the following program: change the number 139 in the DATA statement in Line 5Ø to 179 (the 139 is in bold type); change the number 48 in the DATA statements in Line 5Ø to 237 (the 48 is in bold type); change the number 223 in Lines 152Ø to 155Ø into 247 and the 4496 in Line 6Ø to 4725.

```
10 CLEAR200,30998:DEFUSR0=31000
20 T=0:FORK=0TO43:READN:
   T=T+N:POKEK+31000,N:
   NEXT:READC:IF T<>C THEN PRINT
   "DATA ERROR":END
50 DATA 189, 139, 48,31,3,142,123,12,
   230,192,134,8,183,121,68,79,88,
   73,122,121,68,125,121,23
60 DATA 39,5,88,73,122,121,68,167,
   128,125,121,68,38,233,140,125,
   76,38,221,57,4496
110 CLS:PRINT" SELECT GRAPHICS MODE
   (3-4) ";
120 A$=INKEY$:IF A$<"3"ORA$>"4"
   THEN120
130 T=5-VAL(A$):PRINTA$
140 PRINT" SELECT SCREEN TYPE (0-1) ";
150 A$=INKEY$:IF A$<"0"ORA$>"1"
   THEN 150
160 PRINTA$:ST=VAL(A$)
200 PMODE5-T,1
210 DIMA(14),C1(1),C2(1),C3(1),C4(1)
220 PCLS4:GET(0,0)-(9,5),C4,G
230 PCLS3:GET(0,0)-(9,5),C3,G
240 PCLS2:GET(0,0)-(9,5),C2,G
250 PCLS1:GET(0,0)-(9,5),C1,G
260 PCLS:SCREEN1,ST
270 C=-1:F=3*T-2:GOSUB2070
280 FORX=8TO255 STEP32:C=(C+1)
   AND3:COLORC+1:LINE(X,165)
   -(X+23,188),PSET,BF:NEXT:
   COLOR6-T
290 FOR X=3TO147 STEPT*6:LINE
   (X,3)-(X,147),PSET:NEXT
300 FORY=3TO147 STEP6:LINE(3,Y)
   -(147,Y),PSET:NEXT
310 X=12:Y=12
320 X1=X*6+4:Y1=Y*6+4
330 PUT(X1,Y1)-(X1+5*T-1,Y1+4),
   C1,NOT
340 A$=INKEY$
350 GOSUB 1500
360 IF A$="" THEN 380
370 ON INSTR("PCTRGSMIV",A$) GOSUB
   2200,2500,2300,2800,2600,2700,
   2900,2100,2400
380 IF Y<0 THEN Y=23
390 IF Y>23 THEN Y=0
400 IF X<0 THEN X=24-T
410 IF X>23 THEN X=0
420 GOTO 320
1500 PUT(X1,Y1)-(X1+5*T-1,Y1+4),
   C1,NOT
1510 IFPEEK(338)=191 GOSUB2000
1520 IFPEEK(343)=223 THENX=X-T
1530 IFPEEK(344)=223 THENX=X+T
1540 IFPEEK(341)=223 THENY=Y-1
1550 IFPEEK(342)=223 THENY=Y+1
1560 RETURN
2000 GOSUB4000
2010 P=3*Y+INT(X/8):PX=7-X+
   8*INT(X/8)
2020 IF T=2 THEN VL=F-1 ELSEVL=
   -(F=1)
2030 PK=PEEK(P+VARPTR(A(0)))
2040 PK=PK□AND(255.1-2↑PX):IF T=2
   THENPK=PK□AND
   (255.1-2↑(PX-1))
2050 PK=PK□OR VL*2↑(PX+1-T)
2060 POKEP+VARPTR(A(0)),PK
2070 PUT(216,10)-(239,33),A,PSET
2080 PUT(216,70)-(239,93),A,PRESET
2090 RETURN
2100 RETURN
2200 RETURN
2300 A$=INKEY$:IF A$<>"S"AND
   A$<>"L" THEN2300
2310 IF A$="S" THEN 2330
2320 CLOADM:SCREEN1,ST:RETURN
2330 CSAVEM"",6800,7679,6800
2340 SCREEN1,ST:RETURN
2400 ST=1-ST:SCREEN1,ST: RETURN
2500 A$=INKEY$:IF A$<"0"ORA$>"8"
   THEN 2500
2510 F=((VAL(A$)-1)AND3)+1:RETURN
2600 A$=INKEY$:IF A$<"1" ORA$>"8"
   THEN 2600
2610 J=VAL(A$)-1
2620 GET(J*32+8,165)-(J*32+31,188),A
2630 GOSUB3000:GOTO2070
2700 A$=INKEY$:IFA$<"1" ORA$>"8"
   THEN 2700
2710 J=VAL(A$)-1
2720 PUT(J*32+8,165)-(J*32+31,188),
   A,PSET
2730 RETURN
2800 RETURN
2900 RETURN
3000 CL=F:POKE30999,T-1:N=USR0
   (VARPTR(A(0)))
3010 FORK=0TO23:FORJ=0TO23
   STEPT:F=PEEK(31500+K*24/T+
   J/T)+3-T
3020 X1=J*6+4:Y1=K*6+4
3030 GOSUB4000:NEXTJ,K:F=CL:
   RETURN
4000 ON F GOTO 4010,4020,4030,4040
4010 PUT(X1,Y1)-(X1+5*T-1,
   Y1+4),C1,PSET:RETURN
4020 PUT(X1,Y1)-(X1+5*T-1,
   Y1+4),C2,PSET:RETURN
4030 PUT(X1,Y1)-(X1+5*T-1,
   Y1+4),C3,PSET:RETURN
4040 PUT(X1,Y1)-(X1+5*T-1,
   Y1+4),C4,PSET:RETURN
```

THE SPECTRUM SOUNDS OUT

Spectrum sound effects are limited with the BASIC BEEP command. But with the machine code out you can make sirens and laser zaps—as well as a seven-colour screen border

Normally the microprocessor addresses the computer's memory. And with many home micros if you want to access an outside device—like a printer, a TV or even the computer's own keyboard—you have to do it via a memory location which is tied to an output port. With Z8Ø micros such as the Spectrum, though, you can access the ports directly. It allows you to do this both in BASIC and in machine code via the BASIC commands IN and OUT and assembly language mnemonics **in** and **out**.

WHAT IS A PORT?

A port is the channel of communication between the computer and the outside world, and here the outside world includes the keyboard which is a peripheral as far as the microprocessor and its associated ROM and RAM are concerned.

You have already seen how the BASIC IN command can be used to access a joystick (see page 465) and how the assembly language **in** can be used to access the keyboard (see page 482).

The OUT and **out** commands work in much the same way, only they send data out to the peripherals, rather than taking data in from them. They can be used to control the border of the TV screen, and both commands can be used to output sound through the Spectrum's speaker in a much more adaptable fashion than the BASIC BEEP command. But here, the machine code **out** is under examination, as the BASIC OUT is rather slow.

You need the speed of machine code because of the way that **out** is used to generate sound. What you are doing is to move the speaker in and out. If you do this once, it produces a click, like you sometimes get when you first switch on a record player. But the trick is to do this repeatedly, and very quickly. If the speaker clicks fast enough, a succession of clicks will blur into a low-pitched buzz—if it does it faster still the pitch of the sound will go up.

SOUND **OUT**

The following assembly language routine makes a sound whose pitch increases. Type CLEAR 64599 then enter:



```
      org 64600
      ld a,(23624)
      rrca
      rrca
      rrca
      ld b,0
loop  push bc
      xor 16
      out 254,a
pause nop
      nop
      djnz pause
      pop bc
      djnz loop
      ret
```



It is port 254 that controls the Spectrum's speaker. But the same port also controls the TV border colour. When you **out** a sound you don't want to change the border colour as well, so you have to do a little routine to leave it unaffected.

First, you load the accumulator with the contents of the system variable in memory location 23,624. The Spectrum's colours are specified by a number between 0 and 7. Normally, the bits that control the border colour are bits three, four and five. But when you are controlling it through port 254, it's bits zero, one and two that temporarily change the border colour.

So to make sure that the border colour doesn't change during the sound effect, bits three, four and five have to be shifted three places to the right (into zero, one and two).

SHIFTS AND ROTATES

There are several commands that can do this, but the one selected here is **rrca**—**r**otate **r**ight, with **c**arry, on the **a**ccumulator. This instruction moves all the bits in the accumulator one place to the right. It is called a 'rotate' because the contents of bit zero are moved round and put into bit seven. A simple shift moves each bit one place to the left or right, but fills the spare bit with a zero. Shifts are used in arithmetic operations. You can see that a straight shift to the left multiplies a number by two, and a shift to the right divides by two.

Here, though, a rotate is used because you only care about shifting three of the bits. The contents of the other bits don't matter. The **rrca** also copies the contents of the zero bit into the carry flag. But again, it doesn't matter what the value of the carry flag is, because it is not examined.

To shift the border colour's bits three places to the right, **rrca** is performed three times. This effectively divides the border colour by eight. But then, when the border colour was stored in bits three, four and five the colour (normally a number between 0 and 7) is already multiplied by eight.

So now, when the **out** command is used, it will specify the same border colour as before and no change will be detected.

SETTING THE COUNTERS

The B register is going to be used as a double counter. To start with, it is loaded with 0, and that value is copied onto the stack by **push bc**. The B register cannot be pushed onto the stack by itself. The stack commands, **push** and **pop**, only work on register pairs, so B has to be pushed onto the stack together with the contents of the C register. But as you are not going to do anything with C it won't effect the program.

OUTING THE SOUND

Bit four of port 254 command controls the Spectrum's speaker. By flipping this bit, the

Q+A

How many ports are there?
Theoretically there are 64K possible ports—that is the maximum number that can be addressed by a 16-bit register. But in practice only a few of the possible ports are used. Only one port is used when your Spectrum is in an ordinary hardware configuration—port 254.

With the **in** command you have seen how different parameters feed into the instruction direct the port to different areas of the keyboard (page 482). And in this article, different bits of the machine code **out** command can be used to control the different standard peripheral devices.

However, if your Spectrum is coupled to other non-standard devices other ports can be used. A simple example is when a Kempston joystick is plugged into your Spectrum; port 31 is used to access it (see page 465).

And if you are a dab hand at electronics you could link up your Spectrum to control your central heating system or your electronic synthesizer through different ports.

speaker's diaphragm is sent in and out—creating sound. Bit 4 is flipped by exclusively-oring it with 16. So if bit four is set—that is, it's 1—it is reset to Ø. If it is reset to Ø, it gets set to 1.

The **out 254,a** then **out**s the contents of the accumulator through port 254. (Note that on most commercial assemblers **in** and **out** instructions need brackets round the port number. So if you are not using the assembler published in *INPUT*, this instruction should read **out** (254),**a**.)

SETTING THE PITCH

The instruction **nop** means **n**o **op**eration. It does nothing at all and translates into ØØ in machine code hex. The instruction does take some time to execute though—approximately 1 microsecond, that's a millionth of a second. The reason it is used here, twice, is to slow the processor down as it executes this loop. The speed that the processor goes round the loop controls how fast the speaker's diaphragm goes in and out and, consequently, the pitch.

But as you can see the **nop**s are not just executed twice. The **djnz**—**d**ecrement and **j**ump if **n**ot **z**ero—sends it round and round the **pause** loop executing them many times.

The **djnz** works on the B register. So the first time round the loop it decrements B from Ø to 255. Then it goes round the pause loop a further 255 times until it has decremented to Ø.

Once it has come out of the pause loop the last item is **pop**ped off the stack, back into the BC register pair. This restores the value of the B register, then **djnz** decrements it again and sends the processor back round the **loop** loop again. This **push**es the BC registers back onto the stack. So each time the processor goes round this loop the counter on the stack is decremented and the starting value for the **pause** loop in the B register is one less each time. So the pause is executed one less time and the pitch increases.

SEVEN-COLOUR BORDER

When you specify the BORDER colour in BASIC, you assign it a number between Ø and 7. Then the whole border turns that colour.

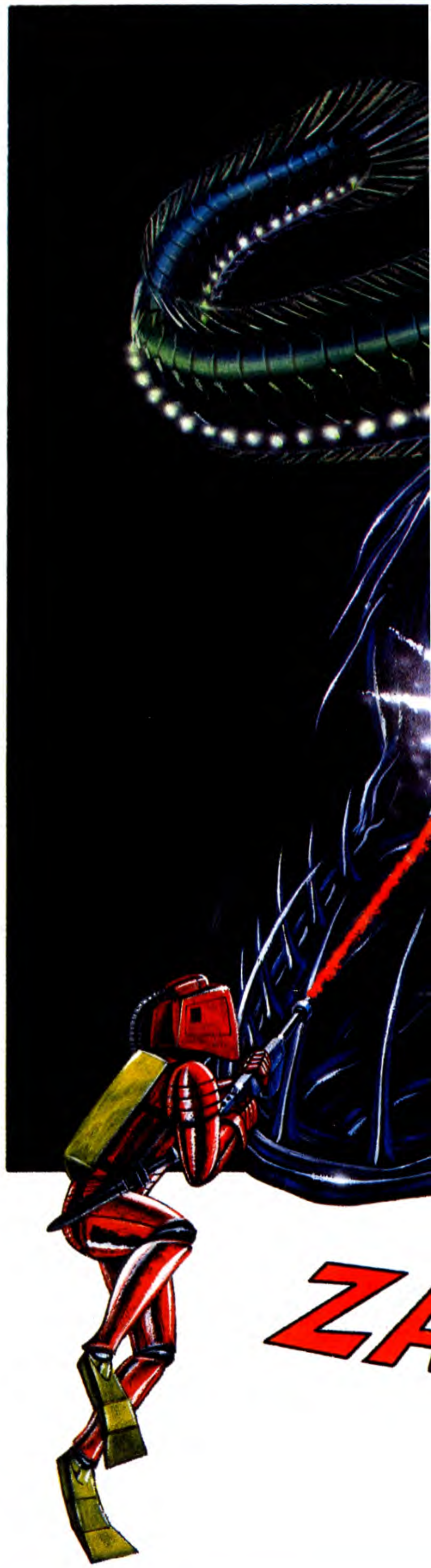
But the following routine uses the machine code **out** command to give a seven-colour border. It could give you an eight-colour border but there is no point in having one of the border colours the same as the screen colour.

```
      org 64600
redo  halt
      xor a
loop  out 254,a
      ld b,205
pause ld e,2
inner dec e
      jr nz,inner
      djnz pause
      ld d,a
      ld a,$7F
      in a,254
      rra
      rts nc
      ld a,d
      inc a
      cp 7
      jr nz,loop
      jr redo
```

SYNCHRONIZING THE SCREEN

The **halt** instruction waits for an interrupt to occur, then sends the processor onto the next instruction. With the Spectrum the interrupt occurs in time with the screen scan. So **halt** starts this routine off when the scan starts at the top of the screen, synchronizing the border bands with the edge of the TV screen.

The **xor a** exclusively-ors the accumulator with itself, which is a quick way to set it to Ø, and the Ø is **out**ed through port 254. Zero, in





P!

machine code as well as in BASIC, means black. So the top part of the screen starts off black.

ADJUSTING THE PAUSE

In this routine the timing is critical. The length of the pause will specify the width of the bands of colour on the border. So the pause here is controlled by two loops—the **inner** loop which sets the delay coarsely and the outer **pause** loop which fine tunes the delay.

The inner loop is performed on the E register, which is set to 2 by **ld e**,2, and decremented by **dec e**. The **jrnz,inner** then makes a relative jump back to the label **inner** if the result is not zero. So this inner loop is performed twice each time the processor goes round the outer loop.

The outer loop is performed 205 times with the B register. It is made up of the **ld b**,2Ø5 which loads the number into the B register and the **djnz** which decrements the B register and jumps if the result is not zero all in one. There is no similar instruction that works with the E register. That's why the decrementing and jump have to be done using two separate instructions.

If you try altering these two values you will see how the value put into the E register alters the width of the bands a lot, while the value put in the B register only alters it a little.

CHECKING FOR BREAK

At some point you will want to escape from this routine. And to do that without switching your computer off you must include an escape routine.

The one used here checks to see if the [BREAK] key has been pressed. This is done using the in instruction.

But first, **ld d,a** loads the contents of the accumulator in the D register. The accumulator is going to be used for something else for a moment and the D register is doing nothing, so it can be used as a temporary store.

The accumulator is then loaded with the hex number 7F. This number specifies the bottom righthand corner of the keyboard (see page 465). Then **in a**,254 takes the bit pattern representing the status of the bottom righthand section of the keyboard into the accumulator.

It is bit zero that denotes the state of the [BREAK] key. If the [BREAK] key is not being pressed, bit zero is 1, but if it is being pressed bit zero is Ø.

A quick way to check this is to rotate bit zero—which is at the extreme righthand end of the register—into the carry flag, then check on the status of the carry flag. The **rra** does the rotation, and the **rts nc** **r**e**t**urn**s** to BASIC when there is **n**o **c**arry. So when [BREAK] is pressed and bit zero is changed from 1 to Ø, the **rts** instruction breaks out of the routine. Otherwise it continues. (Note that the conditional return takes the form **rts** only with the assembler published in *INPUT*. Other Spectrum assemblers use the standard **ret** with conditional returns.)

CLOSING THE LOOPS

Once you have a way out of the routine, the value of the accumulator is restored by **ld a,d**—it just loads the number temporarily

stored in D back into A.

Then A is incremented to specify the next colour. This is compared to the number 7 by **cmp 7**. Seven is the number that specifies white, and white is the colour of the screen.

So if the number in the accumulator is not that specifying white, the **jrnz,loop** jumps back to **out** the colour for the next band. And when this loop has counted up from Ø to 7, the processor moves onto the **jr redo** which takes it right back to the beginning again to wait for the next screen scan.

You'll notice that you don't have to worry about making sounds by accident when changing the border colour. The routine does not affect bit four which moves the speaker. The highest number **out**ed here is 7 and the lowest number that would affect bit four is 16.

SOUND EFFECTS

The time has come to tackle a more complex sound effect using the **out** command. The following routine makes a Star Wars-style laser zapping sound using the combination of a tone of rising pitch and one of falling pitch.

```
        org 646ØØ
        ld a,(23624)
        rrca
        rrca
        rrca
        ld b,Ø
loop    push bc
        xor $1Ø
        out 254,a
        push af
        xor a
        sub b
        ld b,a
        pop af
pausea  nop
        djnz pausea
        xor $1Ø
        out 254,a
        pop bc
        push bc
pauseb  nop
        djnz pauseb
        pop bc
        djnz loop
        ret
```

The first four instructions of this program are exactly the same as the first four in the other sound **out** program. Remember that these preserve the border colour.

The next two—which initialize the two counters, one in the B register and one pushed from the B register onto the stack—are the same as well. And so are the next two which **xor** bit four with 16 and out the result through port 254, though this time the **xor** is suffixed by **$1Ø** which is 16 in hex. This makes the sound.

But then the contents of the accumulator are pushed onto the stack—along with the contents of the flag register. Registers have to be pushed in pairs. The **xor a** clears the accumulator and **sub b** takes the contents of the B register away from Ø—the contents of A—and stores the result in A. This effectively inverses the contents of B. When B is 255, the result of the subtraction is 1. And when B is 1, the result is 255.

The **ld b,a** loads the result of the subtraction back into B. Then the contents of the accumulator which were stored on the stack are **pop**ped off again, back into the accumulator.

The contents of the accumulator are **pop**ped back off the stack. Then there's a pause loop that counts down the contents of the B register—**djnz** decrements the B register and tests for zero, remember. And bit four is flipped and **out**ed to the speaker.

The initial value of the B register is then restored by **pop**ping it off the stack. But this value will be needed again later, so once it has been copied into the B register it is **push**ed back onto the stack. Now it is both in the B register and on the stack.

The next pause loop works on this initial value of B. Then the initial value of the B register is **pop**ped back off the stack again. This has to be done every time after a **djnz** instruction to restore the value of the B register. When the processor comes out of a djnz loop its value must be zero.

The initial value of the B register is then decremented. If it is not zero, the processor loops back to the beginning of the **loop** loop and the whole routine is performed again with a lower value of B. And on the 256th pass, when B is finally decremented to Ø, the processor moves onto **ret** and returns to BASIC.

You will see that the **pausea** loop is performed 256 times on the first pass, once on the second pass and then one more time each time the main loop is executed. But the pauseb loop is performed 256 times on the first pass and one less time for each time the main loop is executed.

Microtip

Making Music on the Spectrum

Using the machine code **out** it is possible to make music on your Spectrum. But adjusting the pause loop to get the pitch you require is laborious.

It would be much easier to write a routine which you could feed two parameters—related to pitch and duration—into. Fortunately, this has already been done for you in the ROM. It is called the BEEP routine and it starts at Ø3F8. And there is an additional subroutine at Ø3B5 called BEEPER which uses the value in HL to control pitch and the value in DE to control the duration.

The pitch value you need to put in HL is given by 437,5ØØ/f—3Ø.125, where f is the note's frequency. Multiply frequency by duration required to give the number you put in DE. There is no 1Ø-second limit on the duration of the note when you are using this method.

If you call the BEEP routine directly the same values for duration and pitch used with the BASIC BEEP should be **push**ed onto the stack.

ADDING NATURAL TONES

The sounds that computers produce usually sound very synthetic because they are too pure. Musical instruments and other devices that generate sound in the outside world tend to be rather haphazard in the way they produce noise. But it is this very randomness that makes the sound pleasing to the human ear. An instrument plays not only the fundamental pure tone but also harmonies of it.

There is a way to add a random factor to your sound effects on the Spectrum, though. The RAM between 16,384 and 32,767 is on chips that are frequently interrupted by the Uncomitted Logic Array which looks after the TV picture and performs other hardware duties.

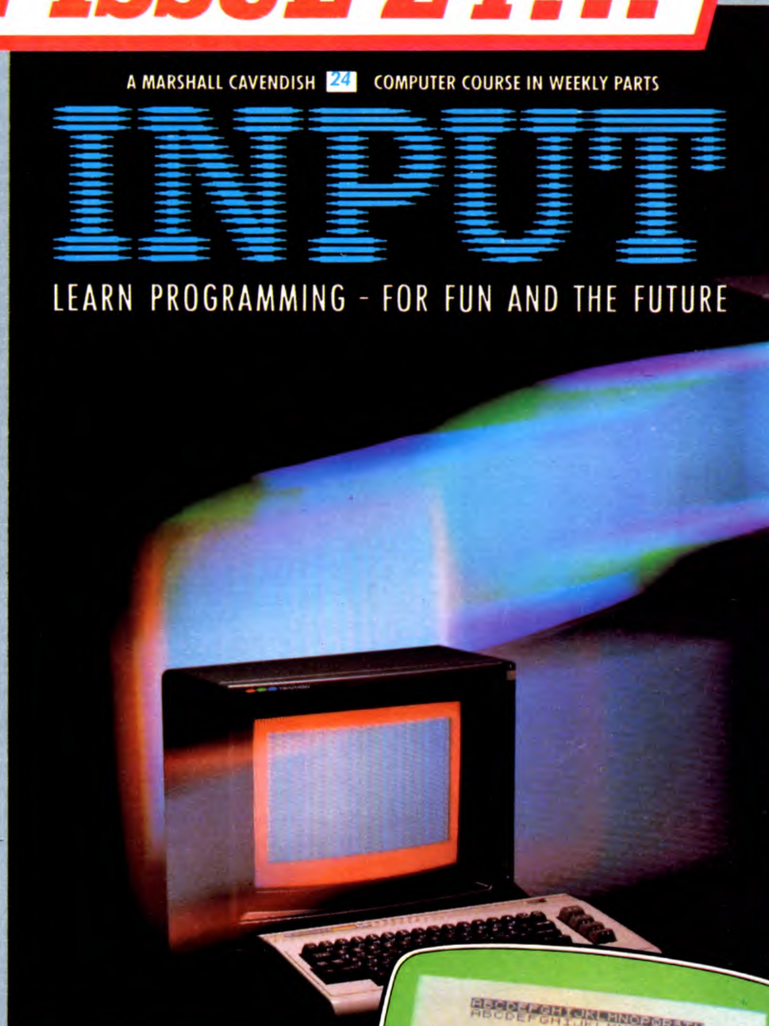
Normally, these interruptions are so quick—typically a couple of microseconds—they are not noticed. But sounds are so time dependent that the ear can sense the slightest variation. So if you put your sound effect programs in this area they will be given a more natural, random feel—and they will last slightly longer.

Unfortunately, the assembler given in *INPUT* occupies this area of memory. But if no printer is being used you can assemble your sound effects programs in the printer buffer, if they are short enough. The printer buffer lies between 23,296 and 23,552. So you can use 23,296 as your origin. And as the printer buffer is protected against overwriting by the BASIC you will not have to do a CLEAR before assembling.

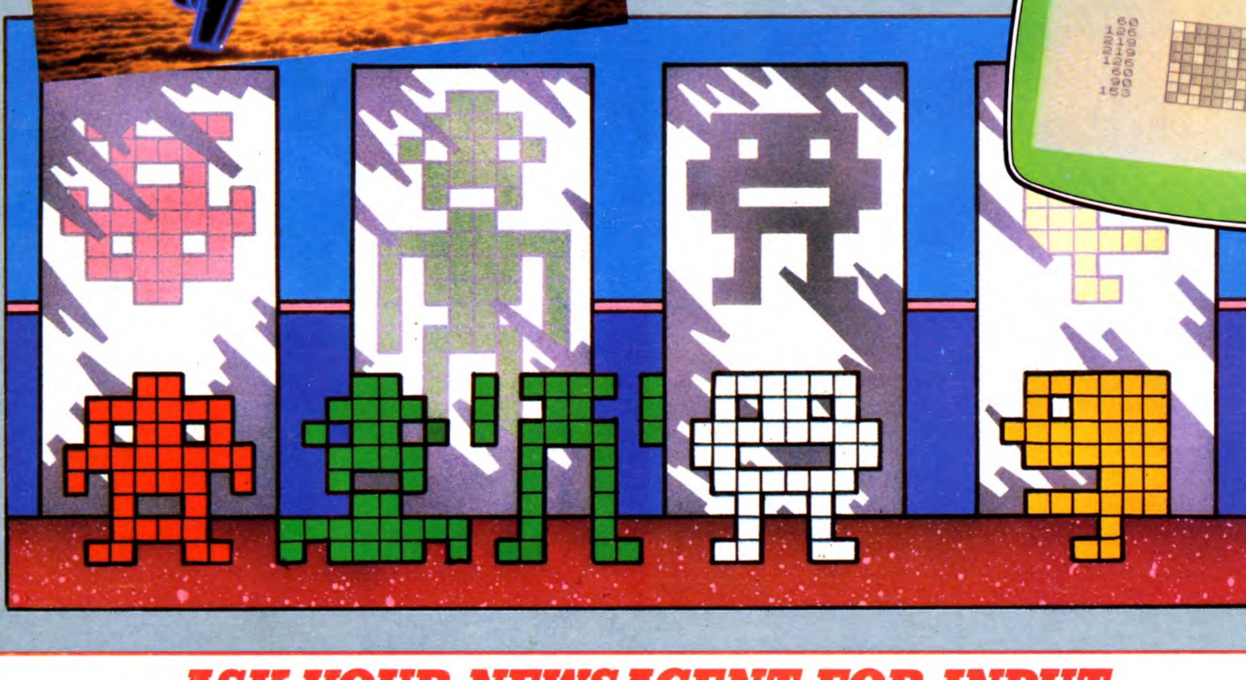
An interim index will be published each week. There will be a complete index in the last issue of *INPUT*.

COMING IN ISSUE 24...

- *Find out the BASIC needed to plot the TRAJECTORIES of flying objects—a principle with many applications*
- *Complete your UDG GENERATOR with new routines to enable quick, easy manipulation of the finished designs*
- *Discover DATABASE MANAGEMENT SYSTEMS, and how to make really good use of your files*
- *Continue the FLIGHT SIMULATOR program by finding out how to set the aeroplane in motion*
- *Plus, a machine code routine for the COMMODORE 64 to give you BASIC control of HI-RES GRAPHICS*









ASK YOUR NEWSAGENT FOR INPUT